



DEFENSE COMMUNICATIONS AGENCY



AD-A166 324

DDN PROTOCOL HANDBOOK

Volume One

DOD MILITARY STANDARD PROTOCOLS

DECEMBER 1985

DTIC
ELECTE
APR 07 1986
S D D

DTIC FILE COPY

DDN

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

86 4 7 040

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Distribution Statement A Approved for public release Distribution unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) <u>NIC 50004</u> NIC 50005; NIC 50006			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION SRI International DDN Network Information Ctr		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION Defense Data Network Program Management Office		
6c. ADDRESS (City, State, and ZIP Code) Menlo Park, CA 94025			7b. ADDRESS (City, State, and ZIP Code) McLean, VA 22102		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
11. TITLE (Include Security Classification) DDN Protocol Handbook (3 vols.) (Unclassified)					
12. PERSONAL AUTHOR(S) Feinler, E.; Jacobsen, O.; Stahl, M.; Ward, G., eds.					
13a. TYPE OF REPORT		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Year, Month, Day) 851200	
15. PAGE COUNT ca. 3000					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Defense Data Network; DDN; DDN protocols; Network protocols; TCP/IP; Transmission Control Protocol/Internet Protocol; Network subscriber access		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The DDN (Defense Data Network) Protocol Handbook is a three volume work which gathers together many documents of interest to those wishing to implement the Department of Defense suite of protocols on various computers to be attached to the DDN. The official Military Standard communication protocols in use on the DDN are included, as are several ARPANET (Advanced Research Projects Agency Network) research protocols which are currently in use, and some protocols currently undergoing review. Tutorial information and auxiliary documents are also included. In addition to its use as a source guide for protocol implementation purposes, the handbook can be used by vendors wishing to make their products compatible with DoD needs, by researchers wishing to improve the protocols, and by implementors of local area networks (LANs) wishing their networks to interact with the DDN.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL E. Redfield			22b. TELEPHONE (Include Area Code) (415) 859-6187		22c. OFFICE SYMBOL EJ 292

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

E. Redfield
SRI International
Network Information Center
333 Ravenswood Ave.
Menlo Park, CA 94025

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED



DEFENSE COMMUNICATIONS AGENCY

DDN PROTOCOL HANDBOOK

Volume One

DOD MILITARY STANDARD PROTOCOLS

DECEMBER 1985

Editors:

Elizabeth J. Feinler

Ole J. Jacobsen

Mary K. Stahl

Carol A. Ward

Additional copies of this document may be obtained from the DDN Network
Information Center, SRI International, 333 Ravenswood Avenue, Room EJ291, Menlo
Park, CA 94025 or from the Defense Technical Information Center (DTIC), Cameron
Station, Alexandria, VA 22304

NTIS

PLEASE NOTE

The 1985 DDN Protocol Handbook is not in itself an official MIL STD. It is a compilation of DoD protocols collected for informational and reference purposes only. Potential contractors who use the material contained in the Handbook to respond to Requests for Proposals (RFPs) or to bid on government contracts, do so at their own risk. Unless this Handbook is specifically cited as the prescribed source to use, we strongly urge that you check with the contracting agency or the Naval Publications and Forms Center (NPFC) to verify that the versions of the MIL STD protocols on which you base your effort are, in fact, the latest versions. The postal address for NPFC is Naval Publications and Forms Center, Code 3015, 5801 Tabor Drive, Philadelphia, PA 19120.

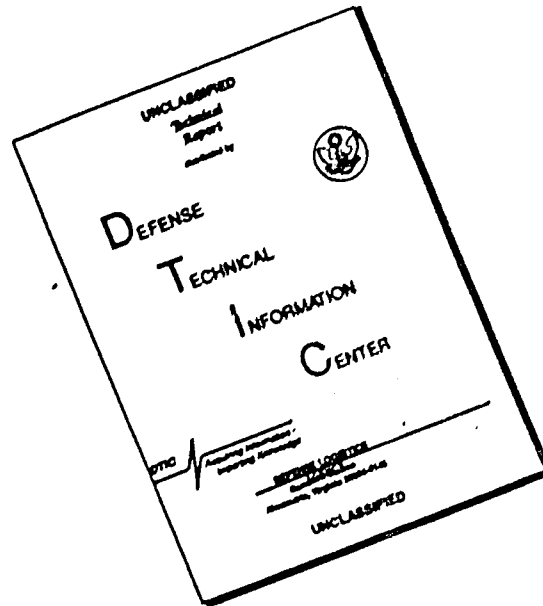
"The Development of Communication Standards in the DoD" by Philip S. Selvaggi. Reprinted from *IEEE Communications Magazine*, Vol. 23, No. 1, January 1985.

DEC-2065 and TOPS-20 are trademarks of Digital Equipment Corporation.

DDN Protocol Handbook. First volume of three-volume set. Printed and bound in the United States of America. Published by the DDN Network Information Center, SRI International, Menlo Park, CA 94025.

Date: December 1985

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

ACKNOWLEDGEMENTS

The DDN Protocol Handbook was compiled by the DDN Network Information Center (NIC) for the Defense Data Network Program Management Office (DDN PMO) of the Defense Communications Agency (DCA) under contract number DCA-200-83-0025.

The editors are indebted to the authors of the many RFCs included in the body of this document. Special thanks goes to the following people for their invaluable support and contributions toward the production of the Handbook: Jonathan B. Postel from the University of Southern California Information Sciences Institute; Michael L. Corrigan, John Claitor, and John R. Walker from the DDN PMO; Edward Brady, Philip S. Selvaggi, Edward A. Cain from the Defense Communications Engineering Center; Chris J. Perry and Michael A. Padlipsky from the Mitre Corporation; and Diane Fountaine from the Office of the Assistant Secretary of Defense for Command, Control, Communications and Intelligence (C³I).

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Avail and/or	
Special	
A-1	



TABLE OF CONTENTS - VOLUME ONE

ACKNOWLEDGEMENTS		iii
SECTION 1: INTRODUCTION TO VOLUME ONE		1-1
SECTION 2: OVERVIEW		1-3
2.1	Purpose of the DDN Protocol Handbook	1-3
2.2	What the Handbook Contains	1-4
2.3	Role of DCA in Protocol Standardization	1-5
2.4	Protocol Review and Acceptance in the DoD	1-7
2.5	Position of DoD on Use of National and International Standards	1-20
SECTION 3: BACKGROUND		1-21
3.1	Brief History of the DDN	1-21
3.2	DoD Architectural Model	1-21
SECTION 4: DDN PROTOCOL CONFIGURATION MANAGEMENT		1-23
4.1	The DDN Program Management Office (DDN PMO)	1-23
4.2	The DDN Configuration Management	1-24
4.2.1	The DDN Configuration Control Group (CCG)	1-24
4.2.2	Blacker Front End Interface Control Document	1-25
4.2.3	OSD Directives	1-41
4.3	Protocol Testing and Validation (ITV&T)	1-59
4.4	Announcement Procedures	1-59
4.4.1	Requests for Comments (RFCs)	1-59
4.4.2	DCA Circulars	1-60
4.4.3	DDN Management Bulletins and Newsletters	1-60
4.4.4	The TACNEWS Service	1-60
SECTION 5: OBTAINING PROTOCOL INFORMATION		1-61
5.1	Military Standards	1-61
5.2	The DDN Protocol Handbook	1-61
5.3	Requests for Comments (RFCs)	1-61
5.4	DDN Management Bulletins	1-61
5.5	NIC Services	1-62
5.6	Other Information Sources	1-63
SECTION 6: DOD MILITARY STANDARD PROTOCOLS		1-65
6.1	Internet Protocol (IP)	MIL-STD 1777 1-67
6.2	Transmission Control Protocol (TCP)	MIL-STD 1778 1-147
6.3	File Transfer Protocol (FTP)	MIL-STD 1780 1-325
6.4	Simple Mail Transfer Protocol (SMTP)	MIL-STD-1781 1-379
6.5	Telnet Protocol and Options	MIL-STD 1782 1-427
6.6	X.25 (Levels 1-3) (Undergoing Review Process)	1-467
6.7	Host Front End Protocol [RFC929] (Undergoing Review Process)	1-523

6.8 Internet Control Message Protocol (ICMP)
(Undergoing Review Process)

1-579

SECTION 7: REFERENCES

1-601

INTRODUCTION

SECTION 1. INTRODUCTION TO VOLUME ONE

Volume One of the three-volume 1985 DDN Protocol Handbook contains an overview of the protocol standardization process and policies within the U.S. Department of Defense (DoD). It discusses the roles of the Defense Communications Agency (DCA) and the DDN Program Management Office (DDN PMO) with respect to this process. Detailed specifications for DoD military standard (MIL STD) computer communication protocols, which are required as part of the protocol suite in use on the Defense Data Network (DDN), are included. The Handbook also outlines the role of the DDN PMO in DDN configuration management, and provides instructions for obtaining additional protocol information.

SECTION 2. OVERVIEW

The 1985 DDN Protocol Handbook is a set of documents which describes specifications for MIL STD communication protocols, experimental protocols, and de facto protocols in use on the DDN and the Defense Advanced Research Projects Agency (DARPA) internet. (The term "internet" is used here to denote the overall topology of the various interconnected networks using the DoD internet protocols). The Handbook includes the official DoD MIL STD communication protocols in use on the DDN, ARPANET research protocols currently in use, and some of the protocols currently undergoing review. Also included are background information, policy information, implementation guidelines, and instructions on how to obtain other protocol information of interest.

Please note that many of the protocols and RFCs that make up the various sections of this Handbook have previously been printed as separate documents. Consequently, some of them have their own separate page numbering. So that the reader can easily distinguish between the two sets of paging, the page numbering for the Handbook as a whole is centered below the footer line, whereas any page numbering specific to an individual document is printed above the footer line.

2.1 Purpose of the DDN Protocol Handbook

The primary purpose of the DDN Protocol Handbook is to serve as a guide for those planning to implement the DoD suite of protocols on various computers to be attached to the DDN, including the ARPANET. For this reason tutorial information and auxiliary documents are included in addition to the protocol specifications themselves. All of this information has been collected into one set of documents that can be used as a source book for implementation purposes.

Military agencies or their contractors who are installing or planning to install Local Area Networks (LANs) or any other type of network which needs to be compatible with the long-haul DDN, will want to consult this Handbook for information and guidelines.

The Handbook provides a means whereby companies and vendors wishing to do business with the DoD can review the protocols required and adapt their products or efforts accordingly. It is to the government's advantage to be able to purchase off-the-shelf products compatible with its computer communications protocols, as these products can be purchased at a considerable cost savings compared to individual custom implementations.

The Handbook is also widely used by researchers in computer science and network communications. Many programmers, computer scientists, communications researchers, and protocol experts have reviewed and/or implemented these protocols and have provided valuable suggestions and feedback to the developers. In the process, they have contributed ideas and improvements which have made the protocols exceptionally robust and noteworthy.

The DoD suite of protocols preceded many of the international computer communication protocols, and so architectural and technical features of the DoD protocols have been incorporated into national and international standards. Members of the various standards bodies have found the Handbook useful for reference and comparison.

2.2 What the Handbook Contains

The 1985 version of the DDN Protocol Handbook updates and obsoletes the 1982 documents entitled *Internet Protocol Transition Workbook*, *Internet Protocol Implementor's Guide*, and *Internet Mail Protocols*, and also the 1983 document entitled, *Internet TELNET Protocol and Options*.

Since 1982, when those documents were issued, many changes have taken place in DoD network communications which necessitate the issuance of this new version of the Handbook. Several protocols which were experimental have now been revised and issued as military standards (MIL STDs). The DDN has been created, and the ARPANET has been split into two networks, the ARPANET and the MILNET. Both the ARPANET and the MILNET are unclassified portions of the DDN; however, each network serves a very different purpose. The MILNET is an unclassified operational military network, whereas the ARPANET remains an experimental network for research and development. These major changes in function and mode of operation of each network are reflected herein by dividing the Handbook into three volumes. Volume One contains the DoD operational MIL STD protocols in use on the DDN. Volume Two contains the current official ARPANET research protocols. Volume Three contains guidelines and auxiliary information useful for implementing either set of protocols.

It is important to note that, at the time of this printing, the MIL STD protocols have essentially the same functionality as the ARPANET protocols. This may change in the future. The MIL STD documents use a formal state machine description in place of the functional description found in the corresponding ARPANET documents. The ARPANET protocols differ slightly from the MIL STDs by offering the implementor more choices of implementation detail than do the MIL STD documents. Otherwise, the protocols are essentially the same.

Several auxiliary documents are included in Volume Three, such as a list of assigned numbers, the ASCII character set definition, guidelines to implementation details, and the like. These provide additional background information or reference data for implementors.

Also included is an overview of the administrative process for the review and acceptance of protocols as military standards by DoD and as ARPANET experimental protocols by DARPA.

2.3 The Role of DCA in Protocol Standardization

Under the Defense Standardization Program, the Defense Communications Agency (DCA) located in Arlington, VA, has been designated the Area Assignee for the development of long-haul communications standards. In this capacity, DCA provides technical guidance and coordinates the efforts of the various DoD participants in the DoD long-haul standards program. Section 2.4 below explains in detail how DCA carries out this assignment and how its efforts are integrated into the overall Defense Standardization and Specification Program (DSSP).

The Development of Communications Standards in the DoD

Philip S. Selvaggi

A detailed overview of procedures in the military standards-making process

IN RECOGNITION of the extreme importance of standardization within the Federal Government, Congress passed the Federal Cataloging and Standardization Act in 1952. Immediately thereafter, the Department of Defense (DoD) established the Defense Standardization Program, with the purpose of meeting the intent of Congress within the Department of Defense. This article briefly describes the aims, policy issues, problems, and philosophy of the Defense Standardization Program, and then elaborates on the workings of the Program with respect to communications standards. The article then goes on to describe the development and methods of coordination for long-haul, tactical, and common communications standards. Following this, brief descriptions of the DoD Operation and Maintenance Standards are provided. The article then continues with a treatment of the DoD Data Protocol Standards Program. In order for data networks to be efficient and reliable, a viable set of protocol standards must be developed for their use. This portion of the article describes the DoD needs in protocol standards, how it has organized to meet these needs, and concludes with a description of future DoD efforts in this area. The article then concludes with a summary of other standardization activities which impact the DoD's standardization programs and activities.

Department of Defense Standardization Program

Introduction

Communications systems designed to serve a wide variety of users very often must procure equipments and facilities from many different sources. To design such systems effectively, a group of viable standards is necessary in order to achieve the DoD objectives of interoperability and required performance levels in a cost-effective manner. Further, when such communications systems are required to interface with other communications systems, standardization is even more necessary because of the need to clearly define the network interfaces, maintain network functionality, and preserve expected customer service. These assertions are supported by the fact that considerable activity is currently underway in national and international standards organizations for the purpose of addressing these issues. On the international scene, the CCITT and ISO have full programs, supported by many meetings throughout the year, which are well attended by representatives of carriers and equipment manufacturers. Standardization groups on the national scene are also very active in many countries. Within the United States, for example, the Electronic Industries Association (EIA) and the American National Standards Institute (ANSI) are fully as active as their international counterparts and their proceedings are just as well attended.

Within the Federal Government, similar pressures have existed for the development and maintenance of a viable set of standards for several decades. These pressures were responsible for the Congressional passage of the Federal Cataloging and Standardization Act in 1952. The objective of this act was to enable the Federal Government to benefit from the many advantages of standardization. The Department of Defense, also interested in benefiting from the positive aspects of standardization, quickly established the Defense Standardization and Specification Program (DSSP) in the same year. The aims and objectives of the DSSP are

basically similar to those of all standardization programs, with one important distinction, however—the DoD must design its communications systems to work reliably in a military environment. Commercial systems, for the most part, work in a relatively benign environment. Hence, there is no need for these latter systems to operate at the levels of performance or to provide the special services that military networks must provide. Because of this fact, commercial standards groups rarely take normal military requirements into account when deliberating over the content of a new standard.

The Defense Standardization and Specification Program

The basic policy regarding DoD standardization is contained in DoD Directive 4120.3, The Defense Standardization and Specification Program. The primary objective of this program is to ensure that optimal materiel standardization is achieved during the design, development, and acquisition process. This objective is achieved by applying standardization principles, such as item commonality, interchangeability, and interface compatibility in engineering and acquisition management. The objectives of the DSSP are achieved by the techniques identified in Table I. If readers desire greater detail on individual aspects of standardization under this program, it may be obtained from [1] and [2].

The DoD standardization program involves the preparation and use of a broad range of equipments, parts, materials, processes, and practices described in specifications, standards, engineering drawings, data item descriptions (DID's), purchase descriptions, and Commercial Item Descriptions (CID's). A guide to the full complement of these documents, the DoD Index of Specifications and Standards (DoDISS), currently lists more than 45000 active standardization documents prepared by DoD activities, other Federal agencies, or industry groups. To support DSSP objectives, more than 7000 standardization projects are either underway or planned. The primary objective for all of this is to achieve a state of materiel standardization within the Department of Defense that will reduce duplicative development and testing costs and control the proliferation of items in the inventory.

TABLE I
Standardization Techniques

- **Developing** standardized products and practices to satisfy military requirements
- **Preparing** standardization documents for engineering and acquisition use of required standardized products and practices
- **Preventing** the preparation of duplicative and overlapping descriptions of materials and services (for example, specifications, standards, purchase descriptions, drawings, and data)
- **Fostering** the reuse of proven technology to satisfy new equipment or system requirements and repetitive use of design features within equipments and systems (inter and intrasystem standardization)
- **Establishing**, as appropriate, uniform types and grades, classes, and sizes of terms and levels of performance requirements which define the characteristics of materiel
- **Developing** methods for systematically reviewing items in the inventory to reduce varieties and sizes to the minimum number compatible with the operating needs of military services

The Defense Standardization and Specification Program is a decentralized program with overall DoD policy, guidance, and administration centered in the Office of the Under Secretary of Defense for Research and Engineering (OUSDRE). Advice and guidance on standardization issues are provided to the OUSDRE by the Defense Materiel Standardization and Specification Board (DMSSB), staffed by Flag Rank/Senior Executive Service representatives from each Department, the Defense Logistics Agency (DLA), and the Office of the Secretary of Defense. Overall management of standardization policies, procedures, and guidance is the responsibility of the Director, Standardization and Acquisition Support within OUSDRE. Day-to-day operations are delegated to the Director, Defense Materiel Specifications and Standards Office (DMSSO). Within each Service and the DLA, a Departmental Standardization Office (DepSO) has been established to manage those portions of the DSSP assigned to the respective Department or Agency.

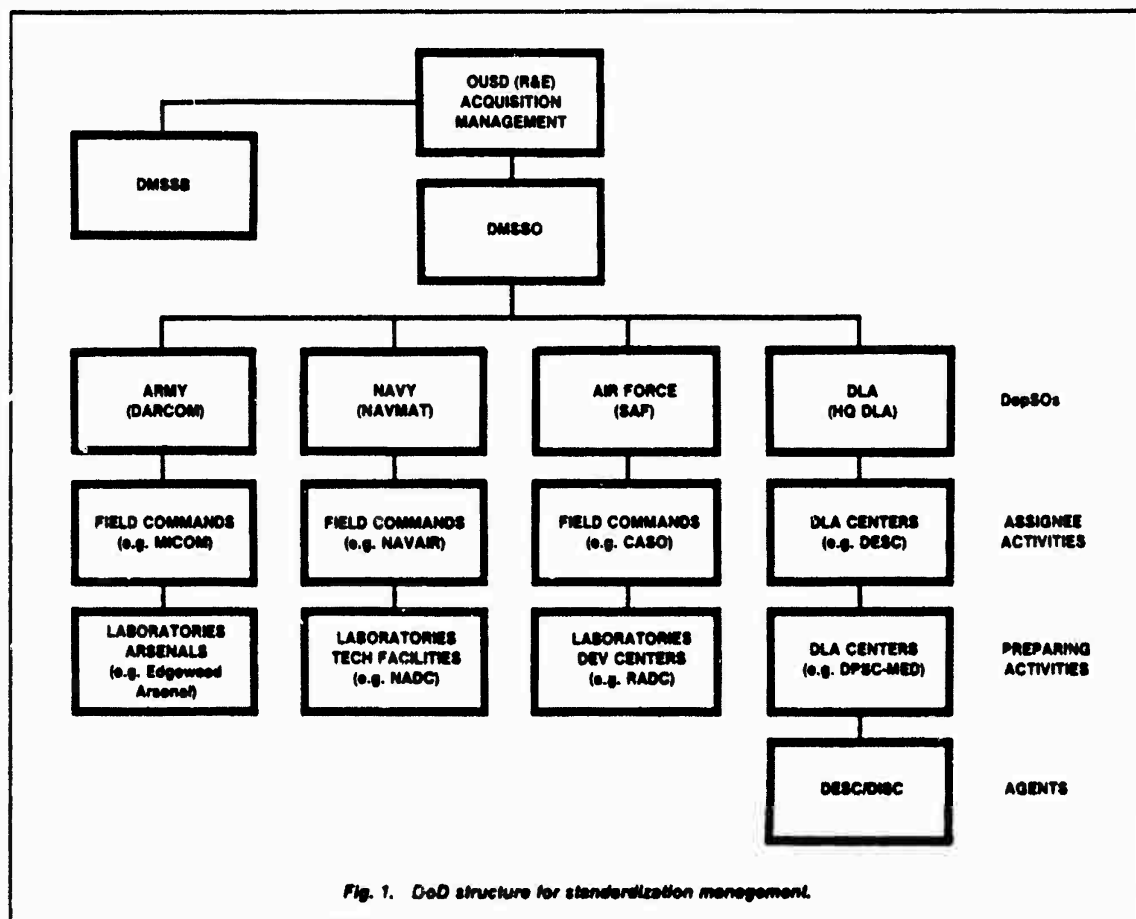
Products used by the military are grouped into logical families, such as space-vehicle components, flight instruments, and land mines. These individual families of products are given the designation of Federal Supply Classes (FSC's). Management and engineering practices, such as reliability, safety, and configuration management, are identified as Standardization Areas. For each FSC and Standardization Area, a military organization or DoD Agency, known as an Assignee Activity (for FSC's) or Lead Service Activity (for Areas), is delegated the responsibility for analyzing, planning for, and ensuring that optimal standardization is achieved. For example, the Defense Communications Agency is currently the Area Assignee for the development of long-haul communications standards for the DoD, while the Army's Communications Electronics Command (CECOM) is the Area Assignee for DoD tactical communications standards.

Development of the actual specifications, standards, and related documents is performed by DoD organizations known as Preparing Activities. It is the Preparing Activity's responsibility to develop, maintain, and coordinate individual DSSP documents, and to ensure that they meet DoD mission requirements. Assignee Activities/Lead Service Activities may also function as Standards-Preparing Activities. Associated with the Assignee/Lead Service Activities and Preparing Activities are several other DoD organizations (for example, participating activities custodians review and user activities, and agents, that assist in the planning, management review, and preparation of FSC Standardization analyses, plans, and standardization documents).

Procedures for preparing and coordinating these documents are outlined in DoD 4120.3-M, Defense Standardization Manual. A simplified organization chart showing the relationship among standardization management offices is shown in Fig. 1. Following is a brief description of the various categories of DoD documents developed under the DSSP.

Standardization Documents

DoD requirements for repetitively produced products are described in specifications while services, practices, processes, and data are described in standards. Both categories of documents have a positive impact on the competitive procurement process. Their primary purpose is to commun-



cate requirements to potential or selected contractors by clearly defining required technical characteristics. These documents are as essential to the acquisition management process as the funds that are used to procure the goods and services.

In order to stimulate maximum competition and contractor creativity, it is DoD policy to use performance-oriented specifications and standards, rather than design specifications, wherever possible. The goal is to indicate what is required of an item rather than how to design, develop, and manufacture the item. In utilizing this approach, there are obviously trade-offs to be made in certain specific situations. Quality, logistics, and standardization considerations sometimes mandate a comprehensive and complex statement of requirements. The DoD strives to maintain a proper balance of these objectives in the specification process. Standardization policies support the philosophy of encouraging the use of performance specifications, but they also recognize that exceptions to this philosophy are often necessary and beneficial.

DoD Standards and Specifications Document Categories

Specifications and standards used by the DoD are divided into three groups—Federal, military, and nongovernment.

Federal specifications, including Commercial Item Descriptions (CID's), cover the very large number of civilian-type products and services used by the Department of Defense. Military specifications and standards describe products and services that are inherently military in nature. The term "nongovernment standards" refers to specifications and standards issued by private sector organizations (not individual companies) which describe goods, services, and engineering practices commonly available to the general public.

A basic order of preference has been established for preparing and using these standardization documents. Where a responsive nongovernment standard is available (or can be made available in time), it shall be used rather than preparing and using a Federal or military document. Where a nongovernment standard does not exist for a commercial product or practice or is unacceptable for DoD's intended application, a CID should be prepared and used. When a more detailed description is required of a commercial product than is permitted in a CID or the item is military-unique, a Federal or military specification or standard should be used. Exceptions to the mandatory requirements for these formal standardization documents are provided in Defense Acquisition Regulation Section 1 Part 12.

Federal Series Documents

Commercial Item Descriptions (CID's) are simplified Federal specifications that describe the key, salient physical and/or functional characteristics of acceptable commercial (or modified commercial) products. These documents are intended to be developed and used when there is reasonable assurance that a satisfactory product from a commercial supplier can be obtained without the need for specifying special design, testing, quality control, or packaging and marking requirements. Where a minimum of special requirements is essential to assure the receipt of an acceptable commercial (or modified commercial) item, these may be included. As the need for greater detail or unique requirements increases, Federal specifications, rather than CID's, are prepared. CID numbers can be recognized by the constant, nonsignificant "A-A-" identifier (for example, A-A-50652, "Life Preserver, Vest, Adult or Child").

Federal Specifications are developed when an acceptable commercially available product or service exists, but specific design, performance, interface, or other essential characteristics cannot be adequately described in a CID. Federal specifications cover material used by, or with the potential for use by, two or more Federal agencies, or new items of potential general applications. Unlike CID's, Federal specifications generally contain a complete description of required items or materials and the provisions for determining compliance with the requirements. Federal specification numbers are made up of two groups of letters, followed by numbers, such as HH-1-524, "Insulation Board, Thermal (Polystyrene)". The first group of letters identifies the commodity and the second represents the first letter of the title.

Federal Standards cover engineering or management processes, practices, or techniques having multiple agency interest. These documents are identified by the indicator "FED-STD-" preceding the document number, for example, FED-STD-4, "Glossary of Fabric Imperfections."

Military Series Documents

Military Specifications are complete descriptions of products which are intrinsically military in character or significantly modified commercial products requiring special features, design, packaging, or quality assurance to satisfy military needs. Military specifications are identified by the prefix "MIL" or "DOD" (for documents in the metric system) followed by the first letter of the first word in the title of the document, (for example, MIL-W-5013, "Wheel and Brake Assemblies, Aircraft").

Military Standards are used to describe engineering and management processes, methods, design criteria, data generating requirements, testing techniques, and definitions. The processes defined by the military standards are of primary concern to DoD activities. These documents are prepared either in book form, indicated by the prefix "MIL-STD-" or "DOD-STD-" (for metric standards) followed by the document number (for example, MIL-STD-965, "Parts Control Program"), or sheet form, identified by "MS" (for example, MS-27423, "Protector-Propeller Shaft Plastic"). MS sheet-form standards are generally used to describe component parts, as opposed to end items, and are being phased out.

Qualified Products Lists (QPL's) are listings of vendor products which were tested to and met previously designated specification requirements. Such preacquisition evaluation is authorized only for products when there is a requirement for special, expensive test equipment not generally available to a potential manufacturer, or when the time to perform testing to assure acceptability of product design, safety, and quality makes it impractical to conduct the tests after contract award. The process by which products are evaluated is called qualification. The fact that a product is listed on a QPL signifies only that, at the time qualification testing was performed, the manufacturer could make an acceptable product. Normal product and quality-assurance testing must still be performed in accordance with specification and contractual terms. QPL's can be authorized for either Federal or military specifications.

Handbooks are reference documents which bring together procedural and technical or design information related to components, equipment processes, practices, and services. A handbook may serve as a supplement to specifications or standards to provide general design and engineering data. Handbooks may be in the Federal or military series.

International Standards are standardization documents which reflect agreements on products, practices, or operations between nations or international associations. It is desirable that such standards be consistent with existing DoD standardization documents to ensure that commitments under these agreements will not adversely affect the DoD design and procurement positions. Similarly, military standards and specifications will often implement treaty commitments, and should be consistent with nontreaty international agreements wherever practical. Examples of international documents include North Atlantic Treaty Organization Standardization Agreements (STANAGS), Quadrilateral Army Standardization Agreements (QSTAG's), and the Air Standardization Coordinating Committee (ASCC) Air Standards.

Nongovernment (Voluntary) Standards are documents prepared by nationally recognized industrial and trade associations and professional societies for use by the general public. These documents are sometimes called voluntary or industry standards. It is DoD policy to make optimum use of such standards when they satisfy military requirements, and to participate with nongovernment organizations in developing the standards.

Nongovernment standards are used by the DoD in three distinct ways: adoption, reference, and excerpts. The generally preferred method is adoption. Adoption is the process by which DoD expresses formal acceptance of nongovernment documents. This also provides visibility to DoD organizational components through incorporation of the adopted standards into the DoD Index of Specifications and Standards (DoDISS). A nongovernment document may also be referenced in a military standardization document. While in most instances it is preferred that these referenced documents be adopted, there are situations where adoption is not appropriate. When only a paragraph or two of a nongovernment document is to be used, it may be more efficient to directly copy or excerpt that portion into the standardization document after permission to do so is obtained from the publishing organization.

It is clear that DoD standardization activities must deal with a vast array of documents and standards categories.

HDBK-412, Site Survey and Facility Design Handbook for Satellite Earth Stations.

Planning Standards—DCS planning standards in the MIL-STD-187 series are expected to provide uniform guidance for the design of the evolving and future DCS. These standards are based on DCS system RDT&E, analyses, and coordinated design decisions. Performance characteristics may meet the criteria for communications system standards, (for example, proven by measured performance) or the criteria for design objectives (that is, best engineering judgment of required performance). An example is MIL-STD-187-320, Transmission Planning Standards for the Defense Communications System.

The following standards have been successfully developed and published as DoD communications standards:

- MIL-STD-188-100—System Technical Standards
- MIL-STD-188-110—Equipment Design Standards for Data Modems
- MIL-STD-188-112—Subsystem Design and Engineering Standards for Cable and Wire
- MIL-STD-188-114—Electrical Characteristics of Digital Interface Circuits
- MIL-STD-188-124—Grounding, Bonding, and Shielding
- MIL-STD-188-140—Equipment Design Standards in Low-Frequency and Lower-Frequency Bands
- MIL-STD-188-161—Design Standards for Facsimile Equipment

Evolution to an all-digital network has raised new standardization issues. In addition to the already published standards, the DoD communications standards program is addressing the following major projects:

- **Common Channel Signaling**—In large commercial networks, an advance in network efficiency and control can be achieved by common-channel signaling. Extensive studies have shown that common-channel signaling can be effective in military networks as well. As a consequence, the DCA is developing such a standard for the DoD long-haul community. At the present time it is envisioned that this standard will be very similar to CCITT #7, the most recent CCITT signaling standard.
- **Timing and Synchronization**—In an all-digital communications system, the timing function affects the complexity of the entire system. The precision of the timing is one factor that determines what functions can be supported by the system, and the manner in which the timing is implemented can have a major impact on survivability. The timing specifications being developed by CCITT differ from those under development by the DCA in the areas of survivability, robustness, security, and functionality. The DoD is therefore developing a timing and synchronization standard which is more appropriate to a military environment.
- **Conversion from Analog to Digital Transmission**—The burgeoning field of digital transmission requires the determination of performance characteristics, as well as establishment of the correlation between user requirements and standards parameters. Several digital transmission projects are in progress; of particular interest is project SLMC 3230, to develop long-haul system transmission performance standards.

- **Integrated Services Digital Network (ISDN)**—The CCITT is investigating the characteristics of this network with respect to future commercial usage. The network will have a large impact upon military communications and standards in general. Investigation into ISDN development has been initiated and will continue for as long as the ISDN remains pertinent to DoD standardization activities.

Tactical Standards Development

Long-haul standards development has its counterpart in the tactical standardization area. The Joint Tactical Command, Control, and Communications Agency (JTC'A) is currently the DoD Lead Service organization for the development of tactical communications standards. Its responsibilities and mechanics of operation are very similar to those in the long-haul arena. It must also develop a yearly Program Plan which governs its activities over a five-year span; the categories of standards treated are also very similar to those in the long-haul area. Needless to say, there is very close coordination between the long-haul and tactical standardization areas in connection with the development of their respective Program Plans.

Common Standards Program

Some years back, under direction from the Joint Chiefs of Staff (JCS), DCA and ECOM (JTC'A's early predecessor as the tactical standardization area assignee) signed a Memorandum of Understanding (MOU) committing both organizations to cooperate in developing common standards for the long-haul and tactical communities. This MOU called for the establishment of a Joint Steering Committee (JSC) to organize and manage this effort. The members of this committee are representatives of services and DoD agencies representing both tactical and long-haul interests. Their objective is to emphasize common standards development and reduce the number of standards that are unique to DoD communications. Such an approach yields many advantages to DoD, not the least of which is that it fosters interoperability between long-haul and tactical communication systems. The level of standards expertise and experience present in this committee, together with the resultant cross-fertilization of ideas, have made the committee a very powerful forum for all DoD communications standards issues which include technical, administrative, and policy matters. The committee is very effective in enabling DoD to discharge its responsibilities in the long-haul and tactical communications areas of the DSSP. The JSC provides the basis for managing communications standards development in the DoD. It establishes the need for new projects and assigns and manages DoD standardization resources. The MOU and JCS action incidentally were the outgrowth of a general realization that there was no real reason for most differences between the requirements in the two standardization areas. Other differences were quickly disappearing as a result of the development of digital communications and microelectronic technology. Since its inception, the common standards program has significantly contributed to interoperability between long-haul and tactical communications systems, and today the bulk of communications standards work in the Department of Defense resides in the common standards category. The development of a common standards program

is therefore necessary to maintain a listing of all such documents for reference purposes, the DoD Index of Specifications and Standards or DoDISS. The DoDISS is a reference publication that lists all military and Federal specifications, standards, handbooks, QPL's, commercial item descriptions, adopted nongovernment standards, international documents, and related publications. Printed editions are provided in alphabetic, numeric, and Federal Supply Classification listings. They are published annually with bimonthly cumulative supplements. Weekly bulletins called DoDISS Notices contain advanced information about selected new and revised DoDISS listed documents.

Communications Standards Development

Long-Haul Standards Development

Under the Defense Standardization Program, the Defense Communications Agency (DCA) located in Arlington, VA, has been designated the Area Assignee for the development of long-haul communications standards. In this capacity, the DCA provides technical guidance to and coordinates the efforts of the various DoD participants in the DoD long-haul standards program. The DCA was created in 1962 for the purpose of integrating DoD long-haul communications requirements, and satisfying these requirements by the establishment of common-user communications systems. To enable it to meet this objective, the DCA has been given the authority to design, build, and manage the required systems. As a consequence, for the past 20 years or so, the DCA has been involved with the engineering planning needed to establish new DoD systems and to update these systems in accordance with the trend of DoD requirements. Its responsibilities make the DCA the ideal body to direct a standards program, since standards are intimately tied in with architectural concepts, interface design, system performance, and equipment and facility design. The standardization process, however, is a broad DoD program involving all DoD elements; to completely meet its standardization responsibilities, the DCA must make extensive use of resources provided by the Military Departments and DoD agencies, strategically supplementing this support with actual in-house engineering resources available at DCA. This arrangement has proven to be extremely effective for the DoD.

The primary management tool for the development of long-haul standards is the Program Plan. This plan, which is issued yearly, covers a span of five years. It contains the status and responsibilities for ongoing projects, a summary of problems facing the DoD standards community, and a technical analysis of future communications planning to identify standardization needs. The development of this plan, as does the development of standards in this area, involves considerable effort. Both responsibilities call for intensive collaborative efforts between the DCA and the Military Departments and Defense Agencies.

Standardization efforts for the Standards for Long-Haul Communications (SLHC) area assignment are currently planned and managed with a view towards promulgation of such standards for the DCS as described in the following categories:

System Design and Engineering Standards—These standards establish overall minimum system and subscriber-to-subscriber measures of performance, for example, the

probability of achieving given delay and/or quality in information transfer. Within circuit and network models, total system and subscriber-to-subscriber performance parameters are apportioned. Included are reference circuits, system diagrams, interrelationships and interconnectivity of component subsystems, and performance goals. The latter includes system efficiency, communications quality, delay, flexibility, availability, reliability, survivability, and security goals. The system design and engineering standards provide the basis for the interconnection of subsystems into a system, as well as the design of individual subsystems, and may address characteristics pertaining to the interconnection of DCS subsystems and interoperability of the DCS with other DoD and non-DoD communications systems. An example is MIL-STD-188-100, Common Long-Haul and Tactical Communication System Technical Standards. Another is MIL-STD-188-323, which provides criteria and guidance for the design of DoD long-haul transmission systems.

Subsystem Design and Engineering Standards—These standards provide electrical performance, message integrity, grade of service, interface standards, and other parameters of subsystems such as switching and transmission subsystems. Where applicable, the parameters for these standards are based on performance parameters derived from the system design and engineering standards. An example is MIL-STD-310A, Subsystem Design and Engineering Standards for Technical Control Facilities.

Equipment Technical Design Standards—These standards provide the minimum electrical performance, such as the dynamic range of operation, input/output parameters, and interface characteristics required of the equipment. Most of these standards are based on the requirements of the appropriate subsystem design and engineering standards. An example is MIL-STD-188-110, Equipment Technical Design Standards Common Long-Haul/Tactical Data Modems.

Interface Standards—These standards specify the required quantitative values for electrical parameters and for operational procedures which affect the transparency of the interface between separate subsystems or systems. The term "transparency" means that

- From the user's standpoint, there should be no additional effort or concern when his communications are traversing separate systems. Ideally, the user's operating procedures should require no change, nor should operations be degraded in any manner.
- Black-box interfaces should be minimized or eliminated by careful analysis of the combined systems being traversed.
- Interface standards provide the necessary technical parameters for the interconnection of the various subsystems of the DCS and are included as an integral part of subsystem and equipment technical design standards in accordance with the system design and engineering standards. An example is MIL-STD-188-114, Electrical Characteristics of Digital Interface Circuits.

Handbooks—Handbooks provide guidance pertaining to the installation of communications equipment facilities. Handbooks cover such areas as theory, equipment layout, signal and power cable runs, environmental control, grounding, transmission lines, and antennas. An example is MIL-

represents a highly significant development in DoD communications standards and has contributed greatly to the furtherance of DoD standardization objectives.

Operational and Maintenance Standards

Communications standards for the DCS are usually thought of as being confined to the MIL-STD-188-series developed under the DSSP. However, another important area of communications standards, vital to the day-to-day operation of the DCS, is the Operations and Maintenance (O&M) standards category. These standards are also developed by DCA, in collaboration with the Military Departments, and published in DCA Circular 300-175-9. They provide criteria by which both the Military Departments and the DCA can gauge how well transmission links and associated systems perform and how well they are being maintained. These criteria take into account degradation due to aging and adverse local environmental conditions. O&M Standards also include DCS Transmission Technical Schedules, which contain itemized listings of all common services provided by the DCS and the circuit parameters required to meet end-to-end performance requirements. The standard also contains a list of carrier offerings which most closely correspond to the defined DCS services. This listing at present pertains only to the CONUS. However, similar listings are currently under preparation for Europe and the Pacific. The Schedules also provide a common language for circuit ordering allocation, activation, operation, and maintenance.

This circular is prepared and updated by an O&M Standards Technical Working Group comprised of representatives from field and operating units. The practice of soliciting direct input from field and operating units started several years ago and resulted in substantial improvements to the document. This was a healthy sign, denoting existing user interest and a serious need for updating the document in order to make it more responsive to actual operating conditions. The result of this progressive step was that yearly meetings have since been held with similar results, and such meetings are planned annually from now on. This Working Group is organized and chaired by a representative from the DCA Standards Office.

An example of this group's effectiveness arose several years ago. The issue addressed by the group had to do with problems in the field when operating modems at 4800 and 9600 b/s over conditioned circuits meeting existing DCS circuit standards. By comparing experiences, the working group members came to the realization that perhaps current DoD transmission schedules were too stringent. To overcome this problem, several new transmission schedules—based upon reducing conditioning requirements—were developed for the DCS. It appeared likely that improved performance could be achieved over DCS data circuits by use of these new schedules. There was, however, insufficient data available from either commercial or government-owned networks which would correlate the effects of reduced circuit conditioning with modem performance at rates greater than 2400 b/s. For this reason, arrangements were made with the Air Force's Rome Air Development Center (RADC) to test appropriate modems over lines simulating the new transmission schedule. Test reports were favorable and the new schedules were validated. Considerable cost savings will be

effected over the years by reducing the need for line-conditioning equipment. These savings would be derived from equipment, installation, and maintenance costs and are directly attributable to the development of O-M standards and strengthening the feedback from field users to the standards development process.

Data Protocol Standards

Protocols are the rules and conventions whereby digital communications are effected between subscribers, between subscribers and networks, and between networks. At present, this category of standards represents an area of extreme interest and activity on the part of the DoD. The DoD has always had an urgent requirement for data communications, and in recent years the critical needs of command and control have magnified this requirement many times over. Emerging computer technology, with a wide range of applicability, has catered admirably to military command and control needs and has provided a sound basis for building computer networks. Although these networks are well suited for meeting military needs, the business and commercial worlds have found similar uses for them; their growth has been rapid in the private sector as well.

It is clear to all, military and nonmilitary users alike, that the successful operation of these new data networks depends upon the development of an effective set of protocol standards. The DoD recognized this fact very early in the game, and for some 15 years has been engaged in a very extensive program of network experimentation concerned with the development of data protocol standards. This research and development effort established the basis for development of the DoD data protocol standards program today, and indeed has provided a stimulus as well for the development of commercial standards in this area. An overriding consideration in the evolution of the DoD program is that it is constrained by the need to meet unique military requirements as well as by the DoD policy to use civil standards where appropriate. In fact, all constraints relating to the DoD role in developing communications standards hold true for its role in developing data protocol standards.

The DoD's Protocol Standardization Program

With DoD's expanding data requirements and technology's growing capability to satisfy these demands, the DoD, in the mid Seventies along with everyone else, found itself in the midst of the data revolution. Requirements technology, programs, and systems were burgeoning everywhere but there were no standards to guide the design and integration of these data systems or to facilitate the transfer of information between data terminals and host computers or between computer networks. In recognition of this situation, the DoD in December 1978 created the DoD Host-to-Host Protocol Standardization Program and directed that the DCA become the Executive Agent for this program. It should be pointed out that, as described earlier, the DSSP had been in existence and functioning well for many years. The Protocol Standardization Program could therefore have been routinely included in the DSSP. The Executive Agent route was chosen, however, because the DoD considered the development of protocol standards to be extremely important and it wished to place strong emphasis on this fact. The duties of

TABLE II
FUNCTIONS OF THE EXECUTIVE AGENT FOR PROTOCOL STANDARDIZATION

- 1) Provide a Forum for discussion with MILDEPS/Agencies on requirements for Standards
- 2) Prepare new proposals for Standards, or enhancements and modifications
- 3) Document Standards and their specifications
- 4) Provide configuration control of Standards and their specifications
- 5) Evaluate, approve, or disapprove proposed Standards and specifications
- 6) Evaluate, approve, or disapprove requests for waivers
- 7) Test proposed Standards and specifications
- 8) Coordinate DoD RDT&E efforts
- 9) Provide a forum for the exchange of technical information
- 10) Monitor Federal, national, and international Standards development
- 11) Ensure that DoD Standards development considers Federal, national, and international Standards
- 12) Justify DoD requirements for waivers from Federal Information Processing Standards (FIPS) PUBS
- 13) Prepare DoD instructions stating DoD policy on protocol standardization
- 14) Coordinate the DoD Protocol Standard Program with corresponding NATO activities

the DCA in conjunction with this assignment are shown in Table II.

DCA's Organization for the Executive Agent Role

The list shown in Table II represents a broad gamut of functions involving managerial, technical, administrative, and policy responsibilities. To handle this full scope of responsibilities the DCA formed three groups, the Protocol Standards Steering Group (PSSG), the Protocols Standards Technical Panel (PSTP), and the Configuration Control Office. The responsibility for implementing protocol configuration control lies in the Defense Communication System Directorate of the DCA. The first two groups have representation from services and DoD agencies, but are chaired by members of DCA. Within DCA, the primary responsibility for each group rests with the DCA organization most concerned with the individual group's function. The relationship between these groups and the Agency is shown in Fig. 2.

The Director, DCA, is actually the Executive Agent for the DoD Protocol Standardization Program. The Chairman of the PSSG acts for the Director with regard to DCA's responsibilities in this area. He recommends standard policy decisions to the Director, and serves as a focal point on his behalf for overseeing the performance of Executive Agent functions. The PSSG itself represents the Military Departments and DoD agencies and serves in an advisory capacity to the DCA.

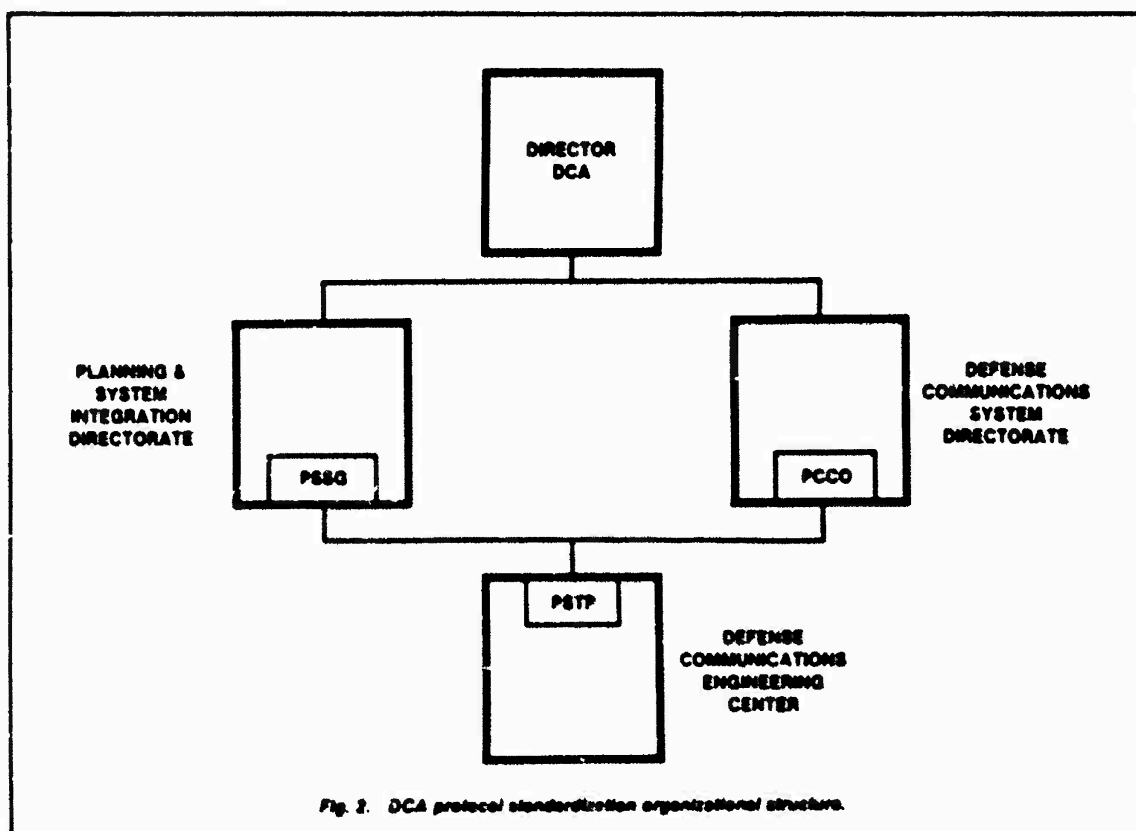


Fig. 2. DCA protocol standardization organizational structure.

TABLE III
MAJOR PROTOCOL STANDARDS STEERING GROUP FUNCTIONS

- 1) Provide a forum for discussion with MILDEPS/Agencies on requirements for Standards
- 2) Coordinate DoD RDT&E efforts
- 3) Monitor Federal, national, and international Standards
- 4) Ensure that DoD Standards development considers Federal, national, and international Standards
- 5) Justify DoD requirements for waivers from FIPS PUBS
- 6) Prepare DoD's stating DoD policy on Protocol Standardization

The responsibility for supporting the PSSG lies in the Planning and System Integration Directorate (PSI) of DCA. This Directorate is responsible for overall DCA planning as well as integration of the various DCA and DoD communications programs. The major functions of the PSSG are summarized in Table III.

The PSSG receives technical support from the Protocol Standards Technical Panel (PSTP). The PSTP reports to the PSSG and is responsible for fulfilling the technical functions of the Executive Agent. Liaison with the PSSG is maintained by having as the PSTP Chairman a member of the PSSG. A summary of its responsibilities is shown in Table IV.

The responsibility for directing the efforts of the PSTP lies in the Defense Communications Engineering Center (DCEC) which, for some time now, has been responsible for the design of the DCS. These responsibilities also include design and planning for the data networks in the DCS as well. For the past several years, the Center has been heavily involved in protocol standards work and has made significant contributions to the DoD protocol standardization effort.

The third area of responsibility is Protocol Configuration Control. The purpose of this function is to ensure interoperability between protocol standards developed by different sources, validate vendor implementations of protocol standards, control deviations and waivers from established standards, and maintain a document data base of approved standards. This function is now discharged by a newly created Configuration Control Office in the Defense Communications Systems Directorate. This office will conduct those activities necessary for establishing configuration control of DoD protocol standards and serves as a focal point for identifying and evaluating requirements for new protocol standards or capabilities. This activity maintains coordination with the PSSG by appointing a representative to this latter group. Table V summarizes the duties involved in the protocol standards configuration control function.

Current DoD Protocol Standardization Plans

By the end of 1978 when the DoD appointed the DCA Executive Agent for the development of protocol standards

TABLE IV
PROTOCOL STANDARDS TECHNICAL PANEL FUNCTIONS

- 1) Prepare new proposals, enhancements, and modifications for Protocol Standards
- 2) Test proposed Protocol Standards and specifications
- 3) Provide a forum for the exchange of technical information

TABLE V
CONFIGURATION CONTROL OFFICE FUNCTIONS

- 1) Document Standards and their specifications
- 2) Provide configuration control of Standards and their specifications
- 3) Evaluate, approve, or disapprove proposed Standards and specifications
- 4) Evaluate, approve, or disapprove requests for waivers

considerable work had been done in developing a Transmission Control Protocol (TCP) and an Internet Protocol (IP) by DCA in conjunction with its design efforts on data-communications networks. The TCP corresponds to the transport layer of the ISO model, which at its inception had provision for an IP. TCP is an end-to-end protocol providing reliable data stream communications with the guaranteed integrity. IP is a protocol for providing data transmission from host-to-host over a set of interconnected diverse networks. At the time these standards were developed, there were no formal standards in these two areas, civil or otherwise. The DoD, however, pressured by urgent requirements and the demands of program implementation, strongly felt the need for such standards in its data communications programs. Therefore, in December 1978 the DoD decreed that these two standards, the TCP and IP, would become official DoD protocol standards. The reason for this action was that both protocols had been devised to meet the essential military requirements of security, survivability, and reliability. These factors were considered to be sufficiently pressing to justify the adoption of the TCP and IP in the absence of comparable commercial standards. Recently, some revisions were made in these standards, but their main substance has not changed since their adoption in 1978. The functions of this TCP are shown in Table VI.

As stated above, an internet protocol was not provided for in the ISO model. However, in the DoD the interconnecting of data networks is an extremely important requirement because of the large number of such networks which serve the DoD. Therefore, early DoD data-communications planning efforts were geared to solving the internet problem. In 1974, for example, the DoD initiated a special study primarily for the purpose of studying DoD data internet requirements. The conclusions of this study played a major role in determining the course of DoD data communications activities over the last eight years. To illustrate the need for this study

TABLE VI
FUNCTIONS OF TCP

- 1) Positive acknowledgment and retransmission
- 2) Check sums on data segments
- 3) Sequencing
- 4) Flow control
- 5) Multiplexing of ULPS
 - Allows many processes within a host to use TCP simultaneously
- 6) Passive and active connections
 - Maintains status information for each data stream
- 7) Precedence and security

TABLE VII
Networks Currently Utilizing DoD IP

General-purpose networks (For example, ARPANET)
Packet radio (SRI, FT BRAGG)
Packet satellite (SATNET, WBNET)
Military Networks (WIN, DoDISS, DOW)
Local Area Networks (ETHERNET, RING, MITREBUS)
Public Data Networks (X.25, IPSS, DATEX-P)
Experimental Networks (RSRE Pilot Packet-Switch Network)

Table VII lists a few of the most important data networks that a DoD common-data-user system must interoperate with.

This list is formidable enough but not nearly complete. The DoD keeps track of the networks it may have to deal with and a recent estimate indicates that the numbers of such nets is considerably in excess of those shown in Table VII. The area of Local Area Networks, for example, has a very great potential for introducing a wide variety of specialized networks into the picture.

Table VIII summarizes the functions of the Internet Protocol Standard. Because the ISO model did not provide for an internet protocol, it is not clear to which layer the IP belongs. There is strong feeling that it belongs somewhere between the 3rd and 4th layers, and recently ISO has placed

TABLE VIII
Functions of IP

1) Addressing
• Provides the network number
• Provides the host address
2) Protocol number
• Keys the next higher-level protocol to be used to interpret internet datagram data
3) Fragmentation and reassembly
• Allows datagrams to be passed through networks with small packet size limits
• Allows reassembly of fragments at destination host
4) Type of service
• Provides a network independent indication of desired quality of service
• Allows user to specify
Precedence
Stream or datagram
Reliability
Speed
Speed over reliability
5) Time to Live
• Specifies maximum time that a datagram is allowed to exist in the internet environment
6) Check Sum
• Provides a check sum for header protection
7) Options
• Source routing
• Record route
• Error report
8) Does not provide
• Flow control
• Data error control
• No acknowledgments of IP datagrams

its function in layer 3, the network layer of the OSI model. At present, the DoD Internet Protocol has been proposed to both ANSI and the ISO through the auspices of the National Bureau of Standards (NBS). The last reports are that it is being well received in both groups. The need for a general internet protocol is well recognized and such a protocol will be developed before long. In addition to the TCP and IP, the DoD has plans for the development of a number of other protocol standards, which are shown in Table IX together with expected completion dates. The development of these standards will be done mainly through contractual support and coordination with the NBS protocol standards program. The DoD has developed a working agreement for the development of protocol standards with the NBS through which it will provide technical assistance to the DoD and also represent the DoD's interests in the appropriate civil standards fora.

National Academy of Science Study

Since the adoption of TCP and IP as formal standards by the DoD, there has been considerable activity in the commercial standards arena with regard to the development of protocol standards. As a result of this activity, both the ISO and CCITT have developed architectural models for protocol standards, and the ISO has developed a transport layer protocol called the Transport Protocol or TP. This TP offers five different classes of operation. The Class-4 category is very similar to the DoD's TCP. The existence of a commercial transport protocol which is different from the DoD standard, however, has raised some problems for the DoD. Concern has been expressed in certain areas of the private sector over their ability to support two different standards, the question has been raised as to whether or not it is cost effective for the DoD to utilize a standard which is different from the commercial one. To resolve these questions, the DoD requested the National Academy of Science (NAS) to evaluate both protocols against DoD requirements and determine whether or not the DoD could use the commercial TP in lieu of TCP.

The NAS has not concluded its study as yet, but it has reached the conclusion that the TCP and the TP are

TABLE IX
DoD Protocol Development Timetable

Protocol	Initial Draft	Final Draft	Published Standard
Message Protocol (MP)	X	X	X
File Transfer Protocol (FTP)	X	X	X
Virtual Terminal	X	X	X
Application Level Protocol (ALP)	FY 85	FY 86	FY 87
Presentation Layer Protocol (PLP)	FY 85	FY 86	FY 86
Session Control Protocol (SCP)	FY 84	FY 85	FY 85
Transmission Control Protocol (TCP)	X	X	X
Internet Protocol (IP)	X	X	X
Gateway-to-Gateway Protocol (GGP)	FY 84	FY 85	FY 85
Datagram Network Interface Protocol (DNIP)	FY 84	FY 85	FY 85
Formal Description Technique	FY 84	FY 85	FY 85

X—Task Completed

TABLE X
Purpose of DoD Protocol Test Laboratories

- 1) **Protocol development**—The process of defining the service and mechanisms, specifications, and implementing the protocol in a high-order language
- 2) **Protocol analysis**—This will provide the capability for interactive and automated "walk-throughs" and special analysis of the protocols to test for such conditions as self-consistency, state reachability, and state "hanging"
- 3) **Implementation testing**—Testing the protocol for conformance to the service specs and how well it performs the services
- 4) **External implementation certification**—Certifying that a user's implementation of a protocol performs correctly against the laboratory's standard implementation
- 5) **Validation and verification**—Procedures for providing the laboratory standard protocol against which other implementations are tested
- 6) **Configuration control**—Providing support for central configuration control of all DoD protocol implementations

functionally equivalent in all respects. No formal recommendations have been formulated as yet and it was expected that the Academy's final report will have been completed in December 1984. This study's conclusion could have far-

reaching effects on the DoD protocol standards program and the implementation of its data networks as well.

Protocol Test Laboratory

An important consideration in the DoD program is protocol testing. To this end, plans have been developed by DCA for a protocol-testing laboratory to be located at the Defense Communications Engineering Center in Reston, VA. The purpose of this test facility is shown in Table X. Figure 3 depicts a preliminary view of the laboratory layout.

The laboratory hardware/software development and implementation will be accomplished in three phases. The implementation of this laboratory will proceed along the lines indicated in the paragraphs below.

Phase 1 will provide: 1) a basic protocol development minicomputer, which will include peripheral storage along with its operating system, editors, language compilers and assemblers, and other basic software tools; 2) three interactive work station terminals (smart terminals) which will allow the protocol developer and tester access to all laboratory functions; 3) two minicomputer "target machines" for use in testing one protocol implementation against another; and 4) a basic coaxial-cable local network for interconnecting all components into an integrated system.

Phase 2 will provide: 1) three additional work-station

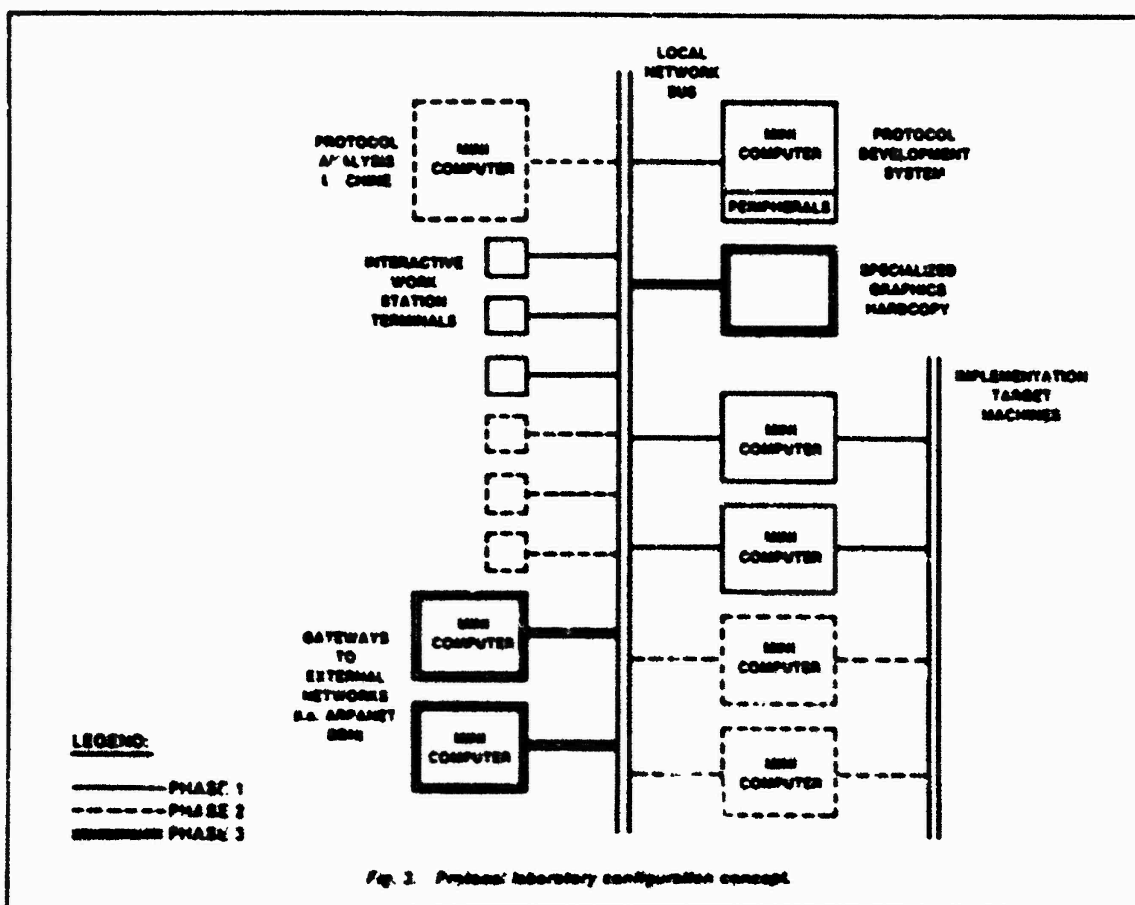


Fig. 3. Protocol laboratory configuration concept.

terminals, 2) a second minicomputer to be used for in-depth protocol analysis provided by special-purpose software packages such as an interactive state machine interpreter, automated test generator, in-depth state machine analysis programs; and 3) two additional target machines to provide the capability for simultaneous protocol testing (testing more than one protocol at the same time).

Phase 3 will procure and implement small minicomputers to provide the function of network gateways to other networks such as the DDN and ARPANET. This will give users the capability to test their protocol implementations running at their own facility against the DoD standard protocol running on one of the target machines in the laboratory. A special graphics terminal with a hard-copy printer may be procured to provide detailed graphic I/O of protocol designs.

Other Standard Activities

NATO Standards Coordination

Another standardization area which requires DCA's participation and activity results from the NATO/United States relationship. Here again, DCA's responsibilities and activities are dictated by public law and DoD policy. Public Law 94-361 decrees that equipment procured for U.S. forces in Europe should be standardized or at least interoperable with equipment used by other members of NATO. This public law is supported by the DoD policy of rationalization with NATO member telecommunications facilities (DoDD 20107), which states that the U.S. shall adhere to U.S.-ratified STANAGS when designing or procuring telecommunications equipment (A STANAG is a NATO Standardization Agreement).

The DCA does not interface directly with NATO on communications standardization matters, but rather works through the U.S. Military Communications-Electronics Board and the Joint Standardization Group for Tactical Command Control and Communications Systems. These two bodies are chartered to provide United States input to NATO standards committees after service and agency coordination. They are responsible, among other things, for providing a U.S. position on NATO Standardization Agreements (STANAGS) in the communications electronics area. A member of the DCA Standards Office staff is a working member of these two groups and the Office generally becomes the focal point for coordinating NATO communications electronics documents within DCA. Once the U.S. has ratified a communications-related STANAG, the DoD policy is that users implement those parameters contained therein which are essential for interoperability. Such action may require either the adoption of a nongovernment standard, the preparation of a new military standard or specification, or the addition of or change to parameters in an existing military standard, all of which actions impact directly upon the Standards Office.

Federal Telecommunications Standards Program

Another important standardization area outside of the DoD is the United States Federal Telecommunications Standards Program. This program, managed by the National Communications System (NCS) Office, has the objective of fostering interoperability of Federal Government telecommunications networks. The NCS achieves this objective largely by monitoring the activities of international and national standardization groups and then adopting their

standards for use in the Federal government. Coordination is maintained with the participating Federal agencies through a Federal Telecommunications Standards Committee Liaison with the DoD standardization program is maintained by having the head of the DCA Standards Office as the DoD representative to this group. Also, the DCA Standards Office has been designated the focal point for coordinating Federal standards within DoD. A critical point here is that the DoD is a very important participant in the Federal Telecommunications Standards Program, and every attempt is made on this program to ensure that Federal standards projects are acceptable to the DoD prior to their being finalized as Federal standards.

Nongovernment Standards Activities of DoD

Nongovernment standards are those standards which are developed outside of the Federal government and are designed primarily for commercial use. This class of standards is extremely important to the DoD because they are used as a basis for the design of commercial equipment and the implementation of communications networks, both on a national and international basis. The DoD has a great need to use such commercial equipment and communications facilities. First, consider the question of communications survivability. The survivability quotient of military networks increases with the number and extent of facilities available. The more easily military networks interoperate with commercial networks, the greater the chances will be for survivability of the nation's communications assets during times of crisis. Second, the DoD very often finds it more economical to lease communications services than to engineer and build new facilities. In fact, the amount of leased services used by the DoD is surprisingly high. Third, the DoD finds it extremely advantageous to avail itself of off-the-shelf commercial equipment. Such an option, in addition to providing the DoD with economies and flexibility in the procurement of equipment, considerably eases the logistics of providing spares, maintenance, and training. The above advantages to the DoD are considerably enhanced if DoD requirements are reflected in the provisions of the commercial standards that guide the design of commercial equipments and networks.

Summary

In summary, the DoD has extensive communications requirements. The implementation of facilities to satisfy these requirements will not be feasible unless these implementations are based upon a sound standards program. Responsiveness to military requirements, interoperability, economy, and performance in a military environment are the prime objectives. The DoD has established the Defense Standardization Program to achieve these objectives. Under this program, the DCA has been assigned the responsibility for developing long-haul communications standards, while the JFCA has the responsibility for developing tactical standards. These two organizations have the joint responsibility to develop common standards to promote interoperability between long-haul and tactical communications.

The DoD's overall policy is to utilize commercial standards wherever possible if such use does not compromise military requirements. To achieve this objective, the DoD actively participates in commercial standards proceedings with

view to introducing the requirements into the commercial standardization process. The DoD also has urgent data communications requirements, and has expended considerable resources on defining these requirements and investigating techniques for satisfying them. It is currently pursuing the development of data protocol standards to complement its efforts in the development of data communications systems. Here again, these standards must take into account the rigors of providing communications in a military environment. The DoD also strongly supports the development of commercial protocol standards and will attempt to use them wherever their use does not compromise essential military requirements. In support of the foregoing, the DoD has established a vital program for the development and test of data protocol standards. The Defense Communications Agency has been designated as Executive Agent for this program. The DoD will coordinate its efforts with the National Bureau of Standards in an attempt to reconcile military standards with commercial standards, and it will make strong efforts to coordinate its activities with NATO as well.

Bibliography

- [1] SD-8 An Overview of the Defense Standardization and Specification Program, May 1 1983.
- [2] DoD 4120.3-M, Defense Standardization and Specification Program Policies, Procedures, and Instructions, Aug 1978 (with changes 1-3).
- [3] DoDD 2010.7, Policy on Rationalization of NATO and NATO Member Telecommunication Facilities, July 6, 1981.
- [4] "A History of the ARPANET: The First Decade," Bolt Beranek and Newman, Inc., April 1, 1981.
- [5] P. Baran, "On Distributed Communications XI Summary Overview," RAND Corp Memo RM-3767-PR 23 p., Aug 1964.
- [6] Department of Defense, "DoD Data Internet Study Phase II Report," Dec 1974.
- [7] V. Cerf, A. McKenzie, R. Scantlebury and H. Zimmerman, "Proposal for an international end-to-end protocol," *ACM Comput Commun Rev*, vol 6 no 1 pp 63-89 Jan 1976.
- [8] Department of Defense DOD 4120.20 "Development and Use of Non-Government Specifications and Standards," Dec 28 1976.
- [9] Richard D. DeLauer, "DoD Policy on Standardization of Host-to-Host Protocols for Data Communications Networks," March 25 1982.
- [10] V. G. Cerf and R. E. Lyons, "Military Requirements for Packet-Switched Networks and Their Implications for Protocol Standardization," *Computer Networks*, vol 7 no 5 Oct 1983.
- [11] Vinton G. Cerf, "The U.S. Department of Defense Internet Architecture," SHAPE Technical Center Symposium on Interoperability of Automated Data Systems, The Hague, Nov 1982.

Philip S. Selvaggi entered the military service in 1943 and was discharged from the United States Air Force in 1946 where he had served as a communications and radar officer. He received his B.E.E. degree from Brooklyn Polytechnic Institute in June 1947 and a Masters Degree in Electrical Engineering from Stevens Institute of Technology in June 1954.

Shortly after graduating from BPI, Mr. Selvaggi joined I.T.T. Labs in Nutley, N.J., where he did design work on PCM digital telephone switching, and missile systems from 1949 to 1956. From 1956 to 1963 he served with the Missile and Surface Radar Division at the RCA Moorestown Plant, where he was manager of the Signal Processing Skill Center which was involved with the design and development of precision tracking radars and advanced development techniques for radar systems.

Mr. Selvaggi left RCA in March 1963 to join NASA's Manned Space Flight Program as assistant director of system integration on the Apollo Program. In June 1964, he joined DCA, where he served as head of the DCS Systems Engineering Directorate until 1974. Since that time, he has served as chief of the Interoperability and Standards Office, primarily involved with developing standards for DoD communications systems. Since 1982, he has become heavily involved in developing DoD protocol standards, and presently serves as chairman of the DoD Protocol Standards Steering Group, which oversees the development of protocol standards for DoD data communications systems.

2.5 Position of DoD on Use of National and International Standards

The Assistant Secretary of Defense for Command, Control, Communications and Intelligence (C³I) stated in an April 1985 memo [1,2] that it is the position of the DoD that whenever international standards are available and can be used to support military requirements, they will be adopted and implemented as rapidly as possible to obtain maximum economic and interoperability benefits. If public or industry standards do not exist to support military requirements for communications protocols, DoD will take the initiative to develop its own standards until such time as suitable national or international standards are available.

SECTION 3. BACKGROUND

3.1 Brief History of the DDN

The ARPANET was the first packet-switching network. This network was designed under a 1969 DARPA research and development program. Initially the ARPANET was an experimental network built to test the concepts of packet switching and resource sharing. As the ARPANET matured, users with operational requirements, rather than experimental requirements, began to use it.

By 1975 there were many operational users of the ARPANET; therefore, responsibility for its operation was transferred to the Defense Communications Agency (DCA).

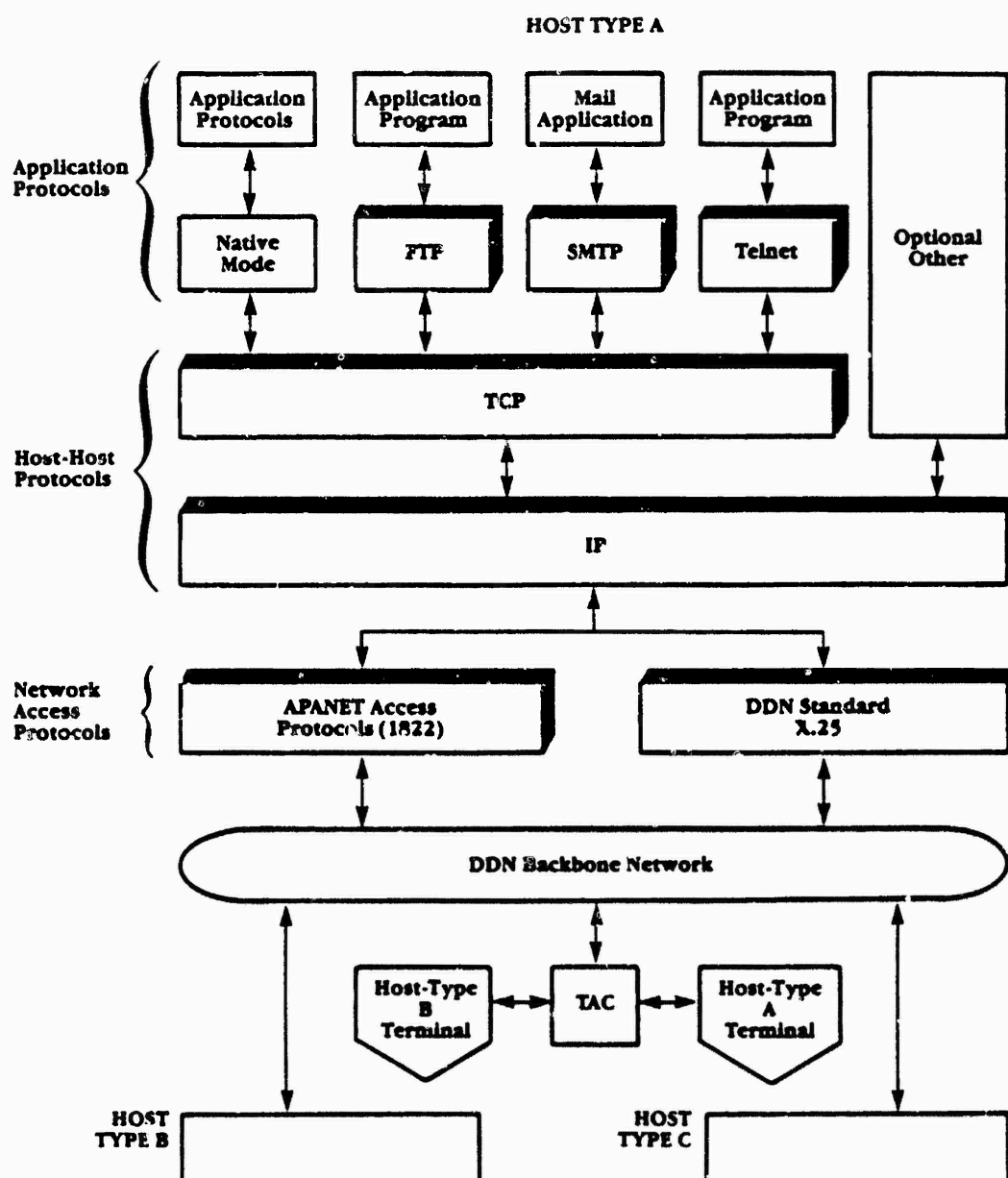
In April 1982, as a result of an intensive internal study of alternative systems, the DoD directed that the Defense Data Network (DDN) be implemented as the DoD common-user data-communications network, based upon ARPANET technology and architecture,

In 1983, DCA divided the ARPANET into two separate networks, the ARPANET and the MILNET, thereby forming the unclassified segment of the DDN.

Both networks are managed by DCA, although the ARPANET remains a DARPA research and development network, while the MILNET and other segments of the DDN are operational military networks. Readers who would like a more detailed overview of the DDN, including a description of the backbone equipment and configuration, should consult the DDN Information Brochure available from the DDN Network Information Center (NIC) or the DDN Program Management Office (DDN PMO).

3.2 DoD Architectural Model

Figure 3-1 shows a graphic representation of the architectural model of the DoD protocol suite. The architecture is similar to, but not identical with, the International Standards Organization (ISO) Open Systems Interconnection (OSI) architecture. [See *Computer Networks*, Vol. 7, No. 5, p. 293-328 (Oct. 1983) for a discussion of the DoD Internet Architecture Model.]



Host and Terminal Interoperability

Figure 3-1: The DoD Protocol Architecture

SECTION 4. PROTOCOL CONFIGURATION MANAGEMENT OF THE DDN

4.1 The DDN Program Management Office (DDN PMO)

The DDN PMO is responsible for the overall management, operations, and policy guidelines for the DDN. It assists new military subscribers in bringing their computers and related equipment onto the DDN. It also manages the ARPANET research and development network on behalf of DARPA.

The DDN PMO provides many services to network users or potential network subscribers. It is responsible for keeping the network up and running, providing users with assistance, setting policies, anticipating growth and expansion, providing configuration management and control, assisting new subscribers with protocol implementation and testing, and advising subscribers on the selection of interface equipment and software.

The DDN PMO also manages the access control and security of the network backbone, and provides liaison to host and node contacts and military sponsors. In addition it provides technical management of contracts for services, equipment, and software obtained from outside corporations and vendors.

There are two major network services on the DDN: the Network Information Center (NIC), provided by SRI International (SRI), Menlo Park, CA, and the Network Monitoring Center (NMC), provided by BBN Communications Corporation (BBNCC), Cambridge, MA. These services are supported under contracts from the DDN PMO.

A third service, DCA Code B641, the Subscriber Requirements and Integration Branch, with representatives from each branch of the military service as well as other sponsored government agencies, assists new network subscribers in assessing their needs for attaching equipment to the network.

DCA Code B650, the Data Operations Division of the DDN PMO, has responsibility for management of all the networks that make up the DDN, including the MILNET and the ARPANET. All operational matters of the DDN should be referred to this office. Code B650 is also responsible for coordinating operational matters within the DDN PMO itself, as well as with other branches and divisions of the DCA.

To provide operational management support for the DDN networks, Code B650 has designated a person at the DDN PMO to act as the primary point of contact (POC) for

operations for each of the DDN networks. Contact the NIC on (800) 235-3155 for the names of these persons.

4.2 DDN Configuration Management

DDN network configuration management is under the control of the Configuration Management Branch, DCA Code B602B.

4.2.1 The Configuration Control Group (CCG)

Matters pertaining to configuration management of the DDN are decided by the Configuration Control Group (CCG). The CCG is chaired by the DDN PMO Technical Manager and is made up of the DDN PMO Division Chiefs. Trouble reports, incident reports, software patch requests, and requests for network configuration changes are reviewed and approved or disapproved by this group.

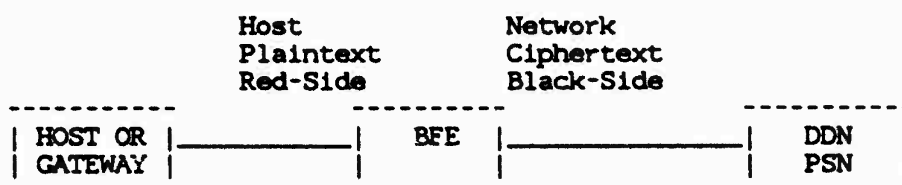
CONFIGURATION MANAGEMENT

6 March 1985

BLACKER BFE ICD

1.0 INTRODUCTION

1.1 The purpose of this document is to define the interfaces of the BLACKER Front Ends (BFE). This document will define the services used on the network or ciphertext side where the BFE interfaces to the Defense Data Network (DDN) and will define the services offered on the host (host or gateway) or plaintext side.



1.2 The BFE acts as a switch (DCE) on an X.25 network. As such the BFE offers an X.25 Standard Service interface to the host. Because of the additional security functionality of the BFE, there are additional requirements on the interface at levels above the X.25 layer. These additional requirements as well as the specific details of the X.25 interface are defined in this document.

1.3 The following terminology will be used in this document. Units of information at the link level will be referred to as frames. Units at the network (X.25 level 3) level will be referred to as packets. Units at the IP level will be referred to as datagrams. Information passed to the actual destination may be referred to as messages with an appropriate adjective as in ICMP message or AMPE message.

1.4 The BLACKER system on DDN uses the X.25 interface as a local interface only. This means that the type of service offered does not provide the end-to-end services of X.25. The BFE will offer a version of DDN X.25 Standard Service as defined in the DDN interface document dated Dec 1983 with the restrictions and modifications defined in this document. This interface will further conform except as described below with FIPS PUB 100 dated 6 July 1983 and CCITT recommendation X.25 dated 1980. The BFE will not provide any support for BBN 1822 interfaces.

1.5 In this document, references to the Administration refer to the Defense Communication Agency.

1.6 All values for fields defined in this document, unless otherwise

BLACKER BFE ICD

6 March 1985

designated, are decimal values. These values will be sent by the binary representation of the decimal value unless otherwise noted.

2.0 HOST/RED-SIDE INTERFACE

2.1 PHYSICAL LEVEL

The BFE will conform to the following specifications:

1. "DEFENSE DATA NETWORK X.25 HOST INTERFACE SPECIFICATION" D.C.A., DECEMBER 1983
2. EIA STANDARD RS-449 NOVEMBER 1977
3. MIL-STD-188-114 MARCH 1976

The BFE will support the signals as listed on Table B-2 of DDN X.25 SPEC. Optional signals supported will be CCITT ID 141 and 142 on the host side.

In RS-449 terms, the BFE will support all Category I circuits in the balanced mode. The BFE will also support all TYPE SR mandatory circuits for synchronous primary channel operation (see RS-449 fig 5.1). The RS-449 37-position connector with a GLENAIR, INC. (or equal) backshell will be used.

The BFE will present a Data Communications Equipment (DCE) interface to the host.

The BFE will operate at speeds from 1.2 to 64 kilobits per second. Data timing will originate at the DCE. RT will supply the data strobe for RD. ST will supply the data request for SD, and TT will supply the data acknowledge/data strobe for SD. This implies that the DCE will control data transfer rates via RT and ST, and the Data Terminal Equipment (DTE) will use ST to generate the data strobe signal, TT. The network DCE supplies timing to the BLACKER DTE and the BLACKER DCE supplies timing to the host DTE. Note: The above signal names RT, RD, etc. are related to the RS-449 names used below.

Only full duplex synchronous operation will be supported.

Interface signal electrical characteristics will be as defined by MIL-STD-188-114. Interface signal functions, directions, and pin assignments will be as defined in RS-449.

The KC-84A is RS-449 compatible in terms of signal conventions.

CONFIGURATION MANAGEMENT

BLACKER BFE ICD

6 March 1985

LISTING OF SIGNALS SUPPORTED BY THE BFE RED SIDE

PIN	RS-449 ABBREVIATION	DCE IS
1	SHIELD	NC
2	SI	+5
3	SPARE	
4	SD	BR
5	ST	BG
6	RD	BG
7	RS	BR
8	RT	BG
9	CS	BG
10	LL	UR
11	DM	BG
12	TR	BR
13	RR	BG
14	RL	IB-
15	IC	-5
16	SF/SR	IB+
17	TT	BR
18	TM	UG
19	SG	CIRCUIT GROUND
20	RC	DCE CIRCUIT GROUND
21	SPARE	
22	SD (see 4)	
23	ST (see 5)	
24	RD (see 6)	
25	RS (see 7)	
26	RT (see 8)	
27	CS (see 9)	
28	IS	IB+
29	DM (see 11)	
30	TR (see 12)	
31	RR (see 13)	
32	SS	IB-
33	SQ	+5
34	NS	IB-
35	TT (see 17)	
36	SB	-5
37	SC	DTE CIRCUIT GROUND

BLACKER BFE ICD

6 March 1985

ABBREVIATIONS OTHER THAN RS-449 SIGNAL NAMES

GND	CHASSIS GROUND
NC	NO CONNECTION
-5	PIN PROVIDES MINUS FIVE VOLTS (MARK, OFF, 1 FOR UNBAL)
+5	PIN PROVIDES FIVE VOLTS (SPACE, ON, 0 FOR BAL OR UNBAL)
IB-	PIN IS OPEN, INTERNAL BIAS OF MINUS FIVE VOLTS
IB+	PIN IS OPEN, INTERNAL BIAS OF FIVE VOLTS
BR	BALANCED RECEIVER
UR	UNBALANCED RECEIVER
BG	BALANCED GENERATOR
UG	UNBALANCED GENERATOR

2.2 LINK LEVEL

The BFE will conform to the following Link Level specifications:

1. "DEFENSE DATA NETWORK X.25 HOST INTERFACE SPECIFICATION", D.C.A., DECEMBER 1983
2. "INTERFACE BETWEEN DATA TERMINAL EQUIPMENT (DTE) AND DATA CIRCUIT TERMINATING EQUIPMENT (DCE) FOR TERMINALS OPERATING IN THE PACKET MODE ON PUBLIC DATA NETWORKS", RECOMMENDATION X.25, CCITT, 1980
3. "COMMUNICATION PRODUCTS HANDBOOK", WESTERN DIGITAL CORP., JUNE 1984

At Level 2, the BFE will use the DDN X.25 High Level Data Link Control, Link Access Procedure-Balanced (HDLC-LAPB) interface protocol.

On the host/red side the BFE will be a DCE.

The interface specified in Bolt Beranek and Newman Inc. (BBN) Report No. 1822 will not be supported.

The HDLC-LAPB interface in the BFE will be implemented using the Western Digital WD2511AN-05 Packet Network Interface chip. NOTE: The Defense Communications Engineering Center (DCEC) certification of this chip is still pending. This chip handles bit oriented, full duplex serial data communications on its Level 1/Level 2 interface side. The computer interface side uses direct memory access.

The "Transparent Modes" offered by the WD2511 chip will not be used.

BLACKER BFE ICD

6 March 1985

2.3 PACKET LEVEL

2.3.1 The BFE will conform to the following Packet Level specifications except as indicated below. Page references are to specification 1. Paragraph references to specification 1 begin with a D (as D2.1.1.1).

1. "DEFENSE DATA NETWORK X.25 HOST INTERFACE SPECIFICATION", D.C.A., DECEMBER 1983
2. "INTERFACE BETWEEN DATA TERMINAL EQUIPMENT (DTE) AND DATA CIRCUIT TERMINATING EQUIPMENT (DCE) FOR TERMINALS OPERATING IN THE PACKET MODE ON PUBLIC DATA NETWORKS", RECOMMENDATION X.25, CCITT, 1980
3. "INTERFACE BETWEEN DATA TERMINAL EQUIPMENT (DTE) AND DATA CIRCUIT TERMINATING EQUIPMENT (DCE) FOR OPERATION WITH PACKET-SWITCHED DATA COMMUNICATIONS NETWORKS", FED-STD 1041; FIPS PUB 100, 6 JULY 1983

2.3.2 (pg.3) Only DDN Standard Service will be offered. No provisions for Basic Service will be made. Any call requests indicating Basic Service will be rejected.

2.3.3 (pg.6) Only physical addressing will be supported. All BFE ports will be assigned a physical address by the Administration. The address will be 12 BCD digits. The address will conform to the format defined in D2.1.1.1 with the following restrictions. The F flag will be 0. All addresses will be 12 BCD digits, sub-addresses will not be supported. Requests for Logical Addressing facilities will receive a CLEAR INDICATION packet with an appropriate diagnostic code (146).

2.3.4 (pg.8) In D2.1.2.1 for the Type of Service facility, Standard Service must be selected on all CALL REQUEST packets. Failure to specify Standard Service will result in a CLEAR INDICATION packet with an appropriate diagnostic code (155).

2.3.5 (pg.10) In D2.1.3, Protocol Identification, DTEs must employ DoD standard IP. The value defined for IP (11001100 binary, CC hex) must be present in the CALL REQUEST packet. Selection of a different value will result in a CLEAR INDICATION packet with an appropriate diagnostic (156).

2.3.6 (pg.11) Negotiated maximum packet size of 1024 octets is strongly recommended in order to allow an IP datagram to fit within a single X.25 packet. Maximum packet sizes of 128, 256, 512, and 1024 octets will be supported by the BFE. A packet size of 1024 is required for hosts accredited to operate at multiple security levels. (See 2.4.4)

BLACKER BFE ICD

6 March 1985

2.3.7 (pg.11) The maximum number of data bits in a complete packet sequence must be no more than 7168 (896 octets). An attempt to send more than 896 octets will result in a CLEAR INDICATION with an appropriate diagnostic code (39).

2.3.8 (pg.A6) The D-bit and Q-bit have no significance to the BFE and are not passed to the destination. These should be set to zero by the DTE.

2.3.9 (pg.A7) There is no support for Logical Addressing. Requests for Logical Addressing facilities will receive a CLEAR INDICATION packet with an appropriate diagnostic code (146).

2.3.10 (pg.A9) BLACKER X.25 addresses are derived from IP addresses as follows:

The IP address is a 32 bit quantity that can be thought of as two parts, the first 8 bits define the network and the remaining 24 bits are network specific. For the BFEs, the 24 bit host field will be mapped to the seven BCD digit host identifier field as follows. The first bit is zero. The next three bits map to the first BCD digit, the next ten bits map to the next three BCD digits and the last ten bits map to the last three BCD digits. The mapping is a value conversion from the binary representation to the BCD representation.

The DDN-RVN will be a class A network. The IP network number for the DDN-RVN is 21. The 24-bit host field will be defined as follows:

0								1								2							
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
+-----+																							
0 PORT								DOMAIN ID								BFE ID							
+-----+																							

The port field will take on the values 0-7, currently defined values are:

0	Attached Host
1	Access Control Module
2	ICMP server

The domain ID and BFE ID fields will only take on the values 000-999.

CONFIGURATION MANAGEMENT

BLACKER BFE ICD

6 March 1985

The DDN-RVN is an X.25 network supporting a version of DDN Standard Service. The X.25 address consists of 12 BCD digits defined as:

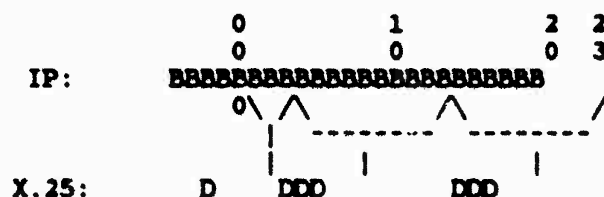
NNNN F DDDDDDD

where

NNNN is a network identifier
F is a flag indicating whether the address is physical or logical
DDDDDD is a network specific address

For the DDN-RVN, NNNN is a value TBD, initially it will be set to 0000. F is 0 indicating physical addressing. DDDDDDD is directly mapped from the host field of the IP address where the first digit is the port ID, the next three digits are the domain ID, and the last three digits are the intradomain BFE ID.

The mapping between the IP host field and the X.25 network specific address is as follows:



2.3.11 INTERRUPT and INTERRUPT CONFIRMATION packets are not supported.

2.3.12 DATAGRAM service is not supported.

2.3.13 There will be no support for PERMANENT VIRTUAL CIRCUITS. All calls will need to be established via CALL REQUEST packets.

2.3.14 Only certain X.25 facilities will be supported.

The following facilities WILL BE supported by the BFE:

Nonstandard default window size	1980 CCITT paragraph 7.1.2
Nonstandard default packet size	7.2.1
Flow control parameter negotiation	7.2.2

BLACKER BFE ICD

6 March 1985

The following facilities WILL NOT BE supported by the BFE:

	1980 CCITT paragraph
Extended packet sequence numbering	7.1.1
Default Throughput Class Assignment	7.1.3
Packet Retransmission	7.1.4
Incoming Calls Barred	7.1.5
Outgoing Calls Barred	7.1.6
One-way logical channel outgoing	7.1.7
One-way logical channel incoming	7.1.8
Closed user group (all varieties)	7.1.9-7.1.15
Reverse charging	7.1.16
Reverse charging acceptance	7.1.17
RPOA selection	7.1.18
Throughput class negotiation	7.2.3
Fast select	7.2.4
Fast select acceptance	7.2.5
D-bit modification	7.2.6
Datagram facilities (all varieties)	7.3

2.3.14.1 For selection of Flow Control Parameters (packet and window size) the BFE will default to a packet size of 128 octets and a window size of 2. A host may select nonstandard defaults for packet size between 128 and 1024 and for window size between 2 and 7. For incoming calls, the host may select whether or not the BFE will negotiate. If the host chooses not to negotiate, the BFE will use the defaults for all calls. If negotiation is selected, the BFE will offer a packet size of 1024 and a window size of 7; the host may then respond with a smaller size if desired.

2.3.15 (Deleted)

2.3.16 DIAGNOSTICS

2.3.16.1 The BFE passes certain diagnostic information back to the host to indicate status information on the communication path and to provide security related information. Diagnostic information is provided when the BFE becomes aware of a reportable event. However, there is no guarantee that the BFE will be able to detect, nor report, all anomalous situations.

2.3.16.2 Diagnostic information is sent in the diagnostic field in X.25 packets. For the X.25 diagnostic codes, the BFE will use the values defined in the CCITT recommendation and in the DDN specification with the following interpretations. DDN code 128, IMP Unavailable, will indicate that the DDN packet switching node to which the BFE is connected is unavailable. DDN code 137, Remote IMP Dead, will indicate that the destination BFE is unreachable.

BLACKER BFE ICD

6 March 1985

2.3.16.3 Diagnostic information related to Emergency Mode status will be passed to the host at the X.25 level. DIAGNOSTIC packets will be sent by the BFE with the following diagnostic codes:

	Code
Entering Emergency Mode	224
Leaving Emergency Mode	225
Emergency Mode Window Open	226

CLEAR INDICATION packets will be sent by the BFE with the following diagnostic codes:

	Code
Call Failed--Address Translation Information Required	227
Call Failed--Emergency Window Open, BFE not in Emergency Mode	228

The host commands the BFE to enter Emergency Mode when the window is opened by using the Emergency Mode Address Facility (2.3.17.3) with the address set to all zeroes.

2.3.17 EMERGENCY MODE

2.3.17.1 One aspect of the BLACKER system operation is the ability to communicate between BFEs in the absence of Access Control Centers (ACC) and/or Key Distribution Centers (KDC). This capability is referred to as Emergency Mode. The use of Emergency Mode involves the host in some of the processing. Depending on a BFE start-up parameter (supplied when the BFE was in communication with an ACC or supplied via BLACKER Variable Carrier (BVC)), when a potential for Emergency Mode exists, either the BFE will automatically enter Emergency Mode and so notify the host, or it will ask the host via the X.25 codes defined in 2.3.16.3. The host must then command the BFE to enter Emergency Mode as defined in 2.3.16.3 and 2.3.17.3 and the BFE will acknowledge.

2.3.17.2 Additionally, since the address translation table described above is maintained by the ACC and in Emergency Mode the BFE may not be able to communicate with the ACC, a host is limited as to what other hosts it may communicate with while in Emergency Mode. The optional X.25 facility defined in 2.3.17.3 allowing a host to provide the address translation information may be used if a host requires flexibility beyond these constraints.

BLACKER BFE ICD

6 March 1985

2.3.17.3 An additional optional user facility will be supported. This facility will allow the DTE to provide the DDN address (black internet address) of the destination BFE (see section 3.5.3). This facility is available only when Emergency Mode is enabled. The facility code is 193. The format is:

```

O  1  1 1 0 0 0 0 0 1
C  2  0 0 0 0 0 1 0 0
T  3      32-bit
E  4      Black
T  5      IP
  6      Address

```

The Black address will be stored with (from diagram in 2.4.2) bits 0-7 in octet 3, bits 8-15 in octet 4, bits 16-23 in octet 5 and bits 24-31 in octet 6. Bit 0 will be the leftmost bit of octet 3, etc. Setting the address field to all zeroes indicates that the BFE should enter Emergency Mode and is sent in response to the BFE message advertising the opening of the Emergency Mode Window (2.3.16.3). If it is necessary to provide the enter Emergency Mode command along with address translation information the facility will appear twice in the CALL REQUEST packet with the enter Emergency Mode command appearing first.

2.3.18 The BFE will support up to 128 simultaneous open logical channels.

2.4 INTERNET PROTOCOL FEATURES

2.4.1 In addition to the X.25 interface, the BFE requires the use of IP as defined in MIL STD 1777. The only restrictions on the use of IP are as follows:

2.4.2 The IP address is a 32-bit value consisting of a network identifier and a network specific host field. There are different formats for this address. The DDN-RVN is a class A network (net number 21) with the following format:

```

      0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
| 0 |   NET   |           HOST           |
+-----+-----+-----+-----+

```

2.4.3 The host connected to a BLACKER Front End will have a Red IP destination address for the datagram. This address will be used by the host to determine the local subnetwork address of the next hop on the Red Virtual Network (RVN), the network comprising all hosts

connected to BFEs. The local subnetwork address will be the X.25 address corresponding to either the actual Red IP destination or the IP address of the gateway that is the next hop to the destination.

2.4.4 The maximum BLACKER IP datagram size is 896 octets. IP datagrams of more than 576 octets should only be sent if there is assurance that the destination is prepared to accept the larger datagrams. An IP datagram must be sent as an X.25 complete packet sequence if the datagram does not fit within a single X.25 packet. A packet or complete packet sequence must contain exactly one complete IP datagram. If IP datagrams are sent in multiple X.25 packets, no more than 32 incomplete datagrams (unfinished packet sequences) may be sent at one time. Only single level hosts are allowed to send datagrams as packet sequences. Hosts accredited to send datagrams at multiple security levels must send and receive datagrams in single packets. Any packet received from such a host with the M bit set will result in the call being CLEARED.

2.4.5 All IP datagrams must contain a security label as defined in the IP security option. This must be the first option on all IP datagrams. The format and values for IP Security Option fields are taken from the "Revised Internet Protocol Security Option (IPSO)". BLACKER currently makes cryptographic distinctions based on the four hierarchical U.S. classification levels and four non-hierarchical National Access Programs as defined in the Revised IP Security Option. In addition, BLACKER is designed to make cryptographic distinctions on up to four additional hierarchical U.S. classification levels and twelve additional Special Access Programs if and when they are specified.

2.4.6 The BFE has a limited amount of space available to buffer datagrams. For datagrams for which authorizations already exist in the BFE, normal flow control procedures will be used. For datagrams which require communication with the ACC, the BFE will only guarantee the buffering of at least five datagrams. These datagrams will be held pending the response from the ACC. Since the receipt of additional datagrams requiring communication with the ACC may overflow the available buffers, such additional datagrams may be discarded.

2.4.7 The BLACKER System supports dual (multiple) homing of hosts by providing authorizations between all BFEs connected to the source and all BFEs connected to the destination hosts. These BFEs each have different network and internet addresses. The source host must provide the mechanism for selecting the communication path from multiple addresses. (BLACKER does NOT support the Dual Homing of its own ACCs or KDCs.)

BLACKER BFE ICD

6 March 1985

2.5 INTERNET CONTROL MESSAGE PROTOCOL FEATURES

2.5.1 The BFE also makes use of ICMP messages to indicate certain information to the host.

2.5.2 The BFE will respond to ICMP ECHO messages with ICMP ECHO REPLY messages.

2.5.3 The BFE passes certain diagnostic information back to the host to indicate status information on the communication path and to provide security related information. Diagnostic information is provided when the BFE becomes aware of a reportable event. However, there is no guarantee that the BFE will be able to detect, nor report, all anomalous situations.

2.5.4 Diagnostic information will be passed in ICMP messages. A DESTINATION UNREACHABLE (type 3) message will be sent when a Request Denied message is received by the BFE from the ACC. Code 1-Host Unreachable will be sent if the Request Denied message indicates that the destination BFE is down. Code 10-Communication with Destination Host Administratively Prohibited will be sent if the Request Denied message indicates that access is denied.

BLACKER BFE ICD

6 March 1985

3.0 NETWORK/BLACK-SIDE INTERFACE

3.1 This section describes the DDN interface of all BLACKER equipment connecting to the DDN.

3.2 PHYSICAL LEVEL

The BFE will conform to the following specifications:

1. "DEFENSE DATA NETWORK X.25 HOST INTERFACE SPECIFICATION" D.C.A., DECEMBER 1983
2. EIA STANDARD RS-449 NOVEMBER 1977
3. MIL-STD-188-114 MARCH 1976

The BFE will support the signals as listed on Table B-2 of DDN X.25 SPEC. No optional signals will be supported on the network side.

In RS-449 terms, the BFE will support all Category I circuits in the balanced mode. The BFE will also support all TYPE SR mandatory circuits for synchronous primary channel operation (see RS-449 fig 5.1). The RS-449 37-position connector with a GLENAIR, INC. (or equal) backshell will be used.

The BFE will present a DTE interface to the network.

The BFE will operate at speeds from 1.2 to 64 kilobits per second. Data timing will originate at the DCE. RT will supply the data strobe for RD. ST will supply the data request for SD, and TT will supply the data acknowledge/data strobe for SD. This implies that the DCE will control data transfer rates via RT and ST, and the DTE will use ST to generate the data strobe signal, TT. The network DCE supplies timing to the BLACKER DTE and the BLACKER DCE supplies timing to the host DTE. Note: The above signal names RT, RD, etc. are related to the RS-449 names used below.

Only full duplex synchronous operation will be supported.

Interface signal electrical characteristics will be as defined by MIL-STD-188-114. Interface signal functions, directions, and pin assignments will be as defined in RS-449.

The KG-84A is RS-449 compatible in terms of signal conventions.

BLACKER BFE ICD

6 March 1985

LISTING OF SIGNALS SUPPORTED BY
THE NETWORK SIDE OF THE BFE

PIN	RS-449 ABBREVIATION	DTE IS
1	SHIELD	GND
2	SI	IB+
3	SPARE	
4	SD	BG
5	ST	BR
6	RD	BR
7	RS	BG
8	RT	BR
9	CS	BR
10	LL	-5
11	DM	BR
12	TR	BG
13	RR	BR
14	RL	-5
15	IC	IB-
16	SE/SR	+5
17	TT	BG
18	TM	IB-
19	SG	CIRCUIT GROUND
20	RC	DCE CIRCUIT GROUND
21	SPARE	
22	SD (see 4)	
23	ST (see 5)	
24	RD (see 6)	
25	RS (see 7)	
26	RT (see 8)	
27	CS (see 9)	
28	IS	+5
29	DM (see 11)	
30	TR (see 12)	
31	RR (see 13)	
32	SS	-5
33	SQ	IB+
34	NS	-5
35	TT (see 17)	
36	SB	IB-
37	SC	DTE CIRCUIT GROUND

ABBREVIATIONS OTHER THAN RS-449 SIGNAL NAMES

GND	CHASSIS GROUND
NC	NO CONNECTION
-5	PIN PROVIDES MINUS FIVE VOLTS (MARK, OFF, 1 FOR UNBAL)
+5	PIN PROVIDES FIVE VOLTS (SPACE, ON, 0 FOR BAL OR UNBAL)
IB	PIN IS OPEN, INTERNAL BIAS OF MINUS FIVE VOLTS

CONFIGURATION MANAGEMENT

BLACKER BFE ICD

6 March 1985

IB+	PIN IS OPEN, INTERNAL BIAS OF FIVE VOLTS
BR	BALANCED RECEIVER
UR	UNBALANCED RECEIVER
BG	BALANCED GENERATOR
UG	UNBALANCED GENERATOR

3.3 LINK LEVEL

The BFE will conform to the following Link Level specifications:

1. "DEFENSE DATA NETWORK X.25 HOST INTERFACE SPECIFICATION", D.C.A., DECEMBER 1983
2. "INTERFACE BETWEEN DATA TERMINAL EQUIPMENT (DTE) AND DATA CIRCUIT TERMINATING EQUIPMENT (DCE) FOR TERMINALS OPERATING IN THE PACKET MODE ON PUBLIC DATA NETWORKS", RECOMMENDATION X.25, CCITT, 1980
3. "COMMUNICATION PRODUCTS HANDBOOK", WESTERN DIGITAL CORP., JUNE 1984

At Level 2, the BFE will use the DDN X.25 High Level Data Link Control, Link Access Procedure-Balanced (HDLC-LAPB) interface protocol.

On the IMP/black side the BFE will be a DTE.

The interface specified in Bolt Beranek and Newman Inc. (BBN) Report No. 1822 will not be supported.

The HDLC-LAPB interface in the BFE will be implemented using the Western Digital WD2511AN-05 Packet Network Interface chip. NOTE: The Defense Communications Engineering Center (DCEC) certification of this chip is still pending. This chip handles bit oriented, full duplex serial data communications on its Level 1/Level 2 interface side. The computer interface side uses direct memory access.

The "Transparent Modes" offered by the WD2511 chip will not be used.

BLACKER BFE ICD

6 March 1985

3.4 PACKET LEVEL

3.4.1 The BFE network interface to DDN conforms to the DDN interface specification dated December 1983.

3.4.2 The BFE operates with a Standard Service interface. It may operate with a Basic Service interface on the network side only.

3.4.3 The BFE is designed to operate with a maximum packet size of 1024 octets but will also operate at 128, 256 or 512 octets. Operating with a maximum packet size of less than 1024 octets may significantly impact performance.

3.4.4 The BFE does NOT make use of INTERRUPT service and does NOT set the D-bit or Q-bit.

3.4.5 For the protocol identification information in the X.25 call, the BFE will use CC hex (11001100 binary) to indicate that IP is the next level protocol and C5 hex (11000101 binary) to indicate the absence of IP and that the next layer of protocol is the encryption layer.

3.5 INTERNET PROTOCOL FEATURES

3.5.1 The BFE will send IP datagrams of up to 1024 octets.

3.5.2 The BFE may not use an IP header on the network side when sending datagrams across a single Black network. This will be indicated in the X.25 protocol field as stated in 3.4.5. This applies ONLY to traffic intended for decryption. Traffic destined for the Black side will always contain an IP header.

3.5.3 The BFE takes the DDN-RVN address (2.3.10) and generates the Black IP address via a table lookup. There is a table in the BFE containing the address translations for the BFE's domain and some interdomain BFE address translation information. This table is maintained by the Access Control Center (ACC) for the BFE. Optionally, information for this table may be provided by the host when the BFE is in Emergency Mode. The Black IP address is passed to the Black side for the Black IP datagram.

3.5.4 The Black X.25 network address is generated from the Black IP address via the algorithm defined for DDN. The BFE will support the full DDN address translation algorithm for both physical and logical addresses.

3.6 INTERNET CONTROL MESSAGE PROTOCOL FEATURES

3.6.1 The BFE will be capable of receiving all ICMP message types and of generating at least ECHO REPLY, PARAMETER PROBLEM, and DESTINATION UNREACHABLE messages.



RESEARCH AND
ENGINEERING

THE UNDER SECRETARY OF DEFENSE
WASHINGTON, D.C. 20301

23 DEC 1978

MEMORANDUM FOR SECRETARY OF THE ARMY
SECRETARY OF THE NAVY
SECRETARY OF THE AIR FORCE
CHAIRMAN, JOINT CHIEFS OF STAFF
DIRECTOR, DEFENSE ADVANCED RESEARCH PROJECTS
AGENCY
DIRECTOR, DEFENSE COMMUNICATIONS AGENCY
DIRECTOR, DEFENSE INTELLIGENCE AGENCY
DIRECTOR, DEFENSE LOGISTICS AGENCY
DIRECTOR, NATIONAL SECURITY AGENCY

SUBJECT: Host-to-Host Protocols for Data Communications
Networks

A number of data communications networks are operating or under development within DoD, without adequate provisions for interoperability. AUTODIN II is expected to become operational during FY 1980, to provide common-user data communications service for DoD computer systems and permit a reduction in the number of specialized data networks. Plans are under way to incorporate within AUTODIN II networks such as the WWMCCS Intercomputer Network (WIN), Intelligence Data Handling System Communications (IDHSC) and the SAC Digital Network (SACDIN), among others. Local networks such as the Community On-Line Intelligence Network System (COLINS) and certain tactical networks must have effective AUTODIN II interfaces.

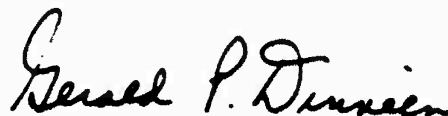
AUTODIN II will provide connectivity for a wide range of systems, but the potential for information exchange beyond narrowly defined communities will be limited without appropriate standards for internet, host-to-host, terminal-to-host and other protocols. As the need to exchange information across network boundaries increases, lack of common protocol standards will become a formidable barrier to interoperability. Techniques in which the protocols of one network are translated into the protocols of another will become increasingly unworkable as the number of protocols and networks requiring interoperation increases.

To insure interoperability of future data networking, I am directing the adoption of a set of DoD standard host-to-host protocols based on the Transmission Control and Internet

Protocols (TCP/IP version 4) developed in the DARPA/DCA internetwork community. DoD requirements for precedence, security, and community of interest controls will be incorporated within the standard protocol set. Use of these protocols will be mandatory for all packet-oriented data networks where there is a potential for host-to-host connectivity across network or subnetwork boundaries. Case-by-case exceptions will be granted only for networks that can be shown to have no future requirements for interoperability. Because the host-to-host protocol being developed for AUTODIN II evolved from an early version of TCP and is unsuitable for internetwork operation, the AUTODIN II TCP will have to be upgraded to the standard protocol set. Recognizing that there may be cost and schedule impacts on the AUTODIN II program, the Defense Communications Agency should perform a cost tradeoff analysis to determine the optimum time for this transition. DCA should provide the results of this analysis by April 1979.

To address these and future protocol issues and promulgate appropriate standards, I am forming an OSD Protocol Standards Working Group chaired by the Director, Information Systems. I ask each addressee to nominate a representative. Names should be provided by 8 January 1979 to LTC Wilcox (695-3287). The first task of this group will be to finalize details of the standard host-to-host protocol set. Draft specifications for these protocols will be available in January 1979. Final specifications should be distributed by April 1979 following review by the working group and testing by DCA and DARPA. At that time, I expect to promulgate these standards and set dates for their adoption.

The Defense Communications Agency is designated as DoD Executive Agent for computer communications protocols and will manage the implementation and development and evolution of standard host-to-host protocols, as designated by the Protocol Standards Working Group. The DCA will forward to this office within 120 days a management plan for carrying out this role.



Gerald P. Dinneen
Principal Deputy



RESEARCH AND
ENGINEERING

THE UNDER SECRETARY OF DEFENSE

WASHINGTON, D.C. 20301

27 MAR 1982

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
CHAIRMAN OF THE JOINT CHIEFS OF STAFF
DIRECTORS OF THE DEFENSE AGENCIES

SUBJECT: DoD Policy on Standardization of Host-to-Host Protocols for Data Communications Networks

- Reference: (a) USDR&E Memo, "Host-to-Host Protocols for Data Communications Networks," 23 Dec 78
(b) DoD Standard Transmission Control Protocol Specification, Jan 80
(c) DoD Standard Internet Protocol Specification, Jan 80
(d) DoD Directive 4120.3, "Department of Defense Standardization Program," 6 June 73
(e) DoDI 4120.20, "Development and use of Non-Government Specifications and Standards," 28 Dec 76

1. The purpose of this memorandum is to clarify DoD policy concerning standardization of host-to-host protocols for data communications networks.

2. The policy cited in reference (a) is reaffirmed, namely: (1) the use of DoD standard host-to-host protocols (Transmission Control Protocol (TCP) and Internet Protocol (IP), references (b) and (c)) is mandatory for all DoD packet-oriented data networks which have a potential for host-to-host connectivity across network or subnetwork boundaries; (2) the Director, Defense Communications Agency, is designated as the Executive Agent for computer communications protocols; and (3) case-by-case exceptions will be granted by the Executive Agent only for networks shown to have no future requirements for interoperability.

3. Reference (a) is not intended to replace the normal DoD standardization procedures established by DoDD 4120.3 (reference (d)). Rather, the Executive Agent function is intended to place increased emphasis and initiative on the important and currently volatile technology of data communications protocol standardization. New standards and modifications to existing standards will be submitted by the Executive Agent to the Defense Department components for ratification and dissemination in accordance with the provisions of reference (d).

4. DoDI 4120.20 (reference (e)) also continues to apply to protocol standards. Thus, it is desired that nongovernment protocol standards be adopted and used in lieu of the development and promulgation of new

documents. Military requirements for interoperability, security, reliability and survivability are sufficiently pressing to have justified the development and adoption of TCP and IP in the absence of satisfactory nongovernment protocol standards. In the future, the Executive Agent will determine whenever unique military requirements justify the development and adoption of unique DoD protocol standards after making every effort to use prevailing nongovernment standards. Moreover, the Executive Agent will make every effort to inject DoD requirements into the nongovernment standard development process through participation in voluntary standards forums and through coordination with other U.S. Government members of such forums. This influence should be exerted with the objectives of both avoiding the need to develop and adopt unique DoD standards and enabling eventual replacement of unique DoD standards with functionally equivalent nongovernment standards.

James P. Wadif
Richard B. Gelauer

CONFIGURATION MANAGEMENT



RESEARCH AND
ENGINEERING

THE UNDER SECRETARY OF DEFENSE

WASHINGTON, D.C. 20301

10 MAR 1983

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
DIRECTORS, DEFENSE AGENCIES
DIRECTOR, JOINT STAFF, OJCS

SUBJECT: Defense Data Network (DDN) Implementation

References: (a) Dep Sec Def Memorandum, Subject: Termination of
AUTODIN II, 2 April 1982
(b) DTACCS Memorandum, Subject: AUTODIN II Phase I
Decision Paper and OSD Guidance for Data Network
Developments, 16 July 1975

This memorandum directs the implementation of the Defense Data Network (DDN) in accordance with Reference (a). This memorandum replaces the previous guidance contained in Reference (b). The Director, Defense Communications Agency (DCA) is overall Program Manager for DDN.

In order to ensure that DDN is implemented as an operationally and economically effective program, the following areas must receive expeditious and comprehensive attention:

- (1) The user system requirements for all DoD data communication systems must be confirmed. This must include accurate operational and technical information.
- (2) System users must select interfacing methods as well as the timeframes required for their systems to connect to the DDN.
- (3) An effective cost recovery scheme which provides for equitable user service costs must be established.

The enclosure hereto contains Guidance and Program Direction applicable to DDN and other DoD Data Networks, and tasking in support of the Defense Data Network Program (to be reviewed by DUSD (C-1) on a continuing basis).

In order to assure success of the DDN, a DDN Coordinating Committee has been established, chaired by the Director of Information Systems with membership from the OJCS, Services, and appropriate Defense Agencies. Intensive and continuing management support from every echelon will be required to make this vital effort a success.

Enclosure

GUIDANCE AND PROGRAM DIRECTION APPLICABLE TO THE DEFENSE DATA NETWORK AND OTHER DoD DATA NETWORKS

- References:
- (a) Dep Sec Def Memorandum, Subject: Termination of AUTODIN II, 2 April 1982
 - (b) DTACCS, Memorandum, Subject: AUTODIN II Phase I Decision Paper and OSD Guidance for Data Network Developments, 16 July 1975
 - (c) DUSD (C3I) Memorandum, Subject: Defense Data Network -- Security Architecture Options, 10 May 1982
 - (d) Director of DCA Memorandum, Subject: Defense Data Network -- Security Architecture Options, 19 Nov 1982
 - (e) Director of NSA Memorandum, Subject: Defense Data Network, 1 Nov 1982
 - (f) USDRE Memorandum, Subject: DoD Policy on Standardization of Host-to-Host Protocols for Data Communications Networks, 23 March 1982

I. Applicability of Program Guidance and Direction

This guidance shall be applicable to the Office of the Secretary of Defense, the Joint Chiefs of Staff, Military Departments, and Defense Agencies. The definition and scope of the Defense Data Network (DDN) will be updated or refined as dictated by changes in user requirements, technological developments, and economic factors. Evolution of the DDN as a Defense Communications System (DCS) element will be governed by the DCS Five Year Plan (FYP) process. Any major changes in the scope, schedules, cost, or composition of the network must be reviewed and approved by DUSD (C3I).

II. Definition of DDN

DDN is a data communications service which will utilize packet technology as its primary switching technique to fulfill the data communications needs of the DoD. The DDN is the data communications service of the Defense Communications System (DCS). The DDN Program Plan, revised 19 May 1982, and augmented by the DDN Security Architecture Reports, (Ref d and e) provides a comprehensive description of the initial planning for the network.

III. Program Strategy for Data Networks

The DDN will supply data communications services in support of critical military operational systems, including WWMCCS and intelligence systems, general purpose ADP and other command based systems and data networks, which have requirements for long-haul data communication services. The DDN will provide connectivity for these subscriber systems with the goal of maximum potential for interoperability.

The DDN is designed to incorporate the maximum practical modularity and flexibility in the backbone system and its various interfaces to accommodate significant changes in user requirements, in ADP and data communications technology, and in the economic factors influencing this program. Contractual and implementation planning for DDN must accommodate variations in the number of switches to be implemented and in the overall implementation schedule of the program. Every attempt must be made to balance this flexibility against reasonable cost impacts to the backbone system and the individual subscriber systems. It is essential that DDN planning be phased in a cohesive total program implementation that is operationally and economically viable.

DUSD (C³I) memorandum, 10 May 1982, (Ref c) directed DCA and NSA to conduct a review of the DDN Security Architecture alternatives for the integration of the various subscriber communities that comprise the DDN. Refs d and e describe the network security architectures that were evaluated.

The approved DDN network security architecture contains two segments, a classified segment and an unclassified segment. The two segments are connected via gates which allow use of the unclassified segment backbone by the classified subscribers. DDN switches in the classified segment (C²I network) are protected to the SECRET level and military encryption devices are employed on all classified segment trunk and access lines. All subscribers on the classified segment are connected to the DDN via the Internet Private Line Interface (IPLI), or equivalent end-to-end encryption (E³) devices. The unclassified segment (MILNET) has switches in restricted locations and uses DES trunk encryption in CONUS, and has switches in SECRET-cleared facilities and uses military encryption devices on OCONUS trunk lines and on OCONUS-CONUS connections. The software in the packet switches and monitoring centers will not be reimplemented, but will be examined for security flaws and brought under strict configuration control. This architecture is referred to in the review as Option 2.2 -- WITH (with IPLIs on all classified hosts and without reimplementing of network software.)

Near-term security for the DDN system will be provided through link encryption of the circuits and segregation of different subscriber communities. Provision of DES link encryption on the MILNET shall proceed as expeditiously as possible, but implementation of systems shall not be delayed solely because such encryption is not in place. Every effort must be made to expedite the development of end-to-end data encryption technology via the Internet Private Line Interface (IPLI) and BLACKER Programs. The focus of these efforts should be to provide host-to-host encryption protection. The BLACKER effort should provide remote key distribution and a trusted (multilevel secure) E³ device suitable for use on the DDN by programs such as the Inter-Service/Agency AMPE, World-Wide Military Command Control Systems (WMMCCS) Information Systems, and SACDIN.

The Director, DCA and all prospective users of the DDN should be fully aware of the requirements of the Privacy Act of 1974, should monitor all follow-on guidance deriving from this Act and related legislation, and should plan for all appropriate changes to the design or operation of their respective systems. The DDN already has design features which provide for "command privacy" and which will assist in minimizing problems from the perspective of "personal privacy."

All DoD data communication systems are required to implement the DoD Standard Host-to-Host Transmission Control and Internet Protocols (TCP/IP) by Ref f. There are ongoing concerted efforts within the government and industry to develop additional standardized data communication protocols. These efforts must be monitored closely to ensure that they meet the functional requirements of the DoD and whenever possible that DoD protocols are in consonance with these efforts.

At the present time, the network access method supported by the DDN is the 1822 interface with the Transmission Control and Internetwork Protocols (TCP/IP). Consistent with our policy of using commercial interface standards whenever possible, DCA is conducting an extensive review in coordination with the National Bureau of Standards of the various options in the X25 network access specification. This review and subsequent testing should result in a specification of the X25 options which will be supported by the DDN. Essential characteristics of this specification will be efficient operation with TCP/IP, with existing 1822/TCP/IP implementations and with the DDN end-to-end encryption capabilities. The wide diversity of incompatible X25 implementations presently available or contemplated in the commercial market could lead to serious operational problems for the DDN and its users. Until the DDN X25 specification has been approved by the DoD Protocol Standards Steering Group, no implementations of X25 will be authorized for use on the DDN.

IV. Guidance for DoD Data Networks

A. Use of the DDN

All DoD ADP systems and data networks requiring data communications services will be provided long-haul and area communications, interconnectivity, and the capability for interoperability by the DDN. Existing systems, systems being expanded and upgraded, and new ADP systems or data networks will become DDN subscribers. All such systems must be registered in the DDN User Requirements Data Base (URDB). Once registered in the URDB, requests by a Service/Agency for an exception to this policy shall be made to DUSD (C³I). Requests for exceptions for joint interest systems shall be routed to DUSD (C³I) through the JCS. Authorization for such special networks may be granted by DUSD (C³I) on the basis of special economic or operational considerations such as:

1. The nature of the data communications services required cannot be satisfied by DDN or a reasonable modification thereto, or
2. Critical operational requirements necessitate immediate implementation actions to provide a data communications service earlier than can be available within the DDN implementation schedule, or
3. The ADP system has time-phased requirements for communications support which can be satisfied and justified, on economic grounds, by an interim network with subsequent transition to DDN when economically feasible.

The DDN Program Manager will, based on the latest information contained in the URDB, prepare projections at several time intervals (e.g., 6 months, one year, two years) of the future topology and data flow characteristics for the networks that comprise the DDN. These projections will be distributed for comment to the OJCS, Services and Agencies. Every attempt will be made in these topology projections to provide equivalent or better service to all current DDN subscribers. Services/Agencies should carefully review these projections and resolve any problems with the DDN Program Manager. Only in case of irresolvable problems should the matter be brought to the attention of the DDN Coordinating Committee.

The DDN Program Manager will provide for informal electronic mail capabilities of the MILNET similar to those presently on the ARPA network. Provisions for funding these services through the Communications Services Industrial Fund (CSIF) should be made available as soon as possible.

Users are encouraged to connect general purpose ADP resources to the DDN for the purpose of sharing computational resources with others of the network. This provision includes the connection of commercially available resources where appropriate.

B. Specific Network Guidance

1. ARPA Network

Those Service/Agency ADP systems that are currently connected to the ARPA network or for which ARPA network connection is planned will form the baseline for the unclassified portion of DDN which has been designated the MILNET. The ARPA network will be partitioned into the MILNET and an Experimental Network as quickly as possible. Electronic mail forwarding capabilities will be provided between the two networks. Positive network access control measures will be implemented on the MILNET and, once fully employed, will allow authorized MILNET user full internet access to the Experimental Network but prohibit full internet access to MILNET from the Experimental Network.

The CONUS switches in the MILNET will be located on restricted access locations and use the DES encryption techniques on all trunks. OCONUS switches will be located in SECRET cleared facilities and military encryption devices will be used on all OCONUS trunks and all OCONUS-CONUS connections.

The Experimental Network (which will retain the name ARPANET network) will be utilized for computer network research and to test concepts to be employed in the DDN. The Experimental Network will be managed and operated by the DDN Program Office. Policies governing its operation will be established by a Steering Committee composed of the DDN Program Manager and sponsors of systems using the Network. The Chairman of this Steering Committee will be appointed by the Director of the Defense Advanced Research Projects Agency.

2. WWMCCS Intercomputer Network

The communications subsystem of the WIN is the basis for the classified portion of the DDN. The DDN will provide service to the WWMCCS ADP community under the direction of the JCS and in accordance with a WIN-DDN Transition Plan to be developed by the DDN Program Manager and the JCS. Department of Defense Intelligence Information Systems and other classified subscriber communities will be added to the WIN communications subsystem to form the C²I network as soon as end-to-end encryption measures are available.

3. Movements Information Network

The USEUCOM Movements Information Network (MINET) will initially be managed as a separate testbed network to determine if urgent transportation requirements of the United States Military in Europe can be satisfied by electronic means. As soon as the MILNET is physically partitioned from the experimental network, the MINET communications subnetwork will become an integral part of the MILNET. Additional users in Europe not covered in the original MINET planning documents will be integrated into the MILNET communications subnetwork by the DDN Program Manager in a manner not to degrade service to the MINET testbed.

V. Tasking in Support of the Defense Data Network Program**A. Tasking for the Chairman, Joint Chiefs of Staff**

1. Revision of various MOPs as required to comply with the guidance contained herein, and publication of a new MOP addressing the DDN.
2. Validate joint-interest user system requirements and forward to DCA.

B. Tasking for the Director, Joint Staff

1. The Joint Staff should monitor the general progress of the tasks identified in this enclosure and assist the DCA, Military Departments, and other Defense Agencies as appropriate.
2. The Joint Staff should continue consideration of the potential requirements of the Unified and Specified Commands which might logically relate to the DDN program. This would include the appropriate potential requirements for NATO interfaces, deployment of switches, interfaces to tactical data systems, changes in the level of survivability needed, and other longer range data communication planning issues.

C. Tasking for the Director, DCA

1. The Director, DCA should accomplish the following tasks and report to DUSD (C³I) as necessary.
 - (a) Develop, operate and manage the DDN on a subscriber-to-subscriber basis.
 - (b) Confirm user system requirements in order to establish and maintain a data base of data communications requirements for system planning and sizing. This action should include both updated projections based on the tasking included in other parts of this enclosure and identification of the specific timeframes when candidate user systems can be connected to the DDN.
 - (c) Develop and refine a reporting format which will allow the Military Departments and Defense Agencies to provide the user requirements data, tasked elsewhere in this enclosure, in a consistent manner.
 - (d) Review the technical concept of operation for each candidate ADP system to ensure that the DDN can adequately support these ADP system requirements.
 - (e) Coordinate with the appropriate agencies to ensure that the DDN specifications properly identify and fully address network security and privacy requirements.

(f) Provide technical review and validation of the protocols, interfaces, precedence, and security features of the DDN and the impacts on user systems. This validation should be accomplished through experimentation, consultation and coordination with the user communities, and evaluation by recognized experts from government and industry.

(g) Develop a network reporting system that provides clear management visibility on network operations of the DDN.

(h) Develop effective cost recovery alternatives for the DDN through the Communications Services Industrial Fund (CSIF) based on equitable rates reflecting actual system usage to the maximum extent feasible.

(i) Establish appropriate management thresholds which will ensure early identification of major changes or problems in the program costs or schedules.

(j) Investigate the potential use of network interfacing devices which will minimize subscriber conversion and operational impacts.

(k) Assist the Military Departments and Defense Agencies in accomplishing their designated tasks.

D. Tasking for the Military Departments and Defense Agencies

1. Develop and forward in a timely manner the required information on all currently operational and planned ADP systems and data networks that require long-haul and area data communications support. This information should be revised as necessary to keep the User Requirements Data Base as accurate as possible.
2. Plan and program to assist the Director, DCA in the implementation of the DDN and user systems.
3. Reassess current concepts of operations and reporting instructions in light of the features and capabilities available through the use of the DDN, and plan for possible improvements.
4. Carefully assess the security features of the DDN and determine how to maximize their security protection. Although these security features may be helpful for ADP system operations, they do not solve the multilevel security problems of the ADP systems.
5. MILDEPs and Agencies are responsible for interfacing their data communications systems to the DDN in accordance with DDN interfacing specification. Where mutually agreed by MILDEPs/Agencies and DCA, DCA will coordinate and manage the development of families of network interfaces.

E. Additional Tasking for the Directors, National Security Agency and Defense Intelligence Agency

Assist the Director, DCA in ensuring the security integrity of the communications systems, including segregation of GENSER-SI traffic, segregation of subscriber communities, Defense Switched Network (AUTOVON) dial-up circuit protection procedures, overall network security, and other appropriate areas of security.



RESEARCH AND
ENGINEERING

(C3I)

THE UNDER SECRETARY OF DEFENSE

WASHINGTON DC 20301

14 MAY 1984

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
DIRECTORS OF THE DEFENSE AGENCIES
DIRECTOR, JOINT STAFF, OJCS

SUBJECT: DoD Policy on Defense Data Network (DDN) Protocols

Under Secretary of Defense (Research and Engineering) Memorandum "Defense Data Network (DDN) Implementation," dated 10 March 1983 prohibited the use of X.25 connections to the DDN until the DDN X.25 specification had been approved by the DoD Protocol Standards Steering Group (PSSG). The DDN X.25 specification has been approved by the PSSG and it is hereby authorized for use on the DDN.

The requirement to use the DoD Standard Host-to-Host Transmission Control and Internet Protocols (TCP/IP) promulgated by USDRM Memorandum, "DoD Policy on Standardization of Host-to-Host Protocols for Data Communications Networks," 23 March 1982, remains in effect.

With the approval of the DDN X.25 specification in the Defense Communications Agency (P&SI) Memorandum, "DDN X.25 Specification," 4 January 1984, the DoD has taken another step toward standardized data communication protocols, along with industry and the international community. Since the use and availability of the X.25 protocol is widespread, every effort should be made to bring DoD in line with all the data communications users in this area. The DDN X.25 will be used as the access protocol; TCP and IP will continue to be used in the internet and transport layers. The 1822 protocol will continue to be supported by the DDN until phased out via evolution.

All new systems and systems undergoing major redesign are required to use levels 2 and 3 of the DDN standard X.25 protocol for interfacing to the DDN. Case-by-case waivers will be considered by Assistant Secretary of Defense (C3I). The on-going I-S/A AMPE and BLACKER programs are specifically requested to take steps to implement the DDN Standard X.25 specification as early as possible.

James P. Wade, Jr.
Assistant Secretary of Defense
(C3I)

WASHINGTON, D.C. 20301-2040

-4 17:00 05 08 59

8 APR 1985

SUBJECT: National Research Council Report on Transport Protocols for Data Networks

[illegible]

In order to insure that DoD is in a posture to evaluate TP once it is in wider use in the commercial sector, request you initiate the following actions:

- (1) develop the DoD military requirement specification for TP to insure that industry is aware of DoD needs as TP is commercially implemented.
- (2) insure that appropriate advisory representation is provided to commercial standards working groups that are currently refining TP under the auspices of the National Bureau of Standards.
- (3) insure that the DCA protocol test facility can accommodate TP testing as required when commercial implementations are available.
- (4) develop a transition strategy for Option 2 of the report to include estimated resource requirements.
- (5) evaluate the detailed recommendations presented in the Report (pages 61-64) as they apply to Option 2.

Original and enclosure
sent to Co. 202 for
release and appropriate
action.

David C. Patton

cc: NBS, Mr. Bob Blanc

4.3 Protocol Testing and Validation (IVV&T)

Protocol testing of vendor products and host implementations are carried out by DCA Code B613, the Development Test and Evaluation Branch of the DDN PMO. Vendors wishing to have products validated should contact this branch for information. Emphasis of this group to date has been on testing backbone equipment and software. Test procedures are being developed to assist site personnel with testing and validating implementations of the DoD internet protocols on their local hosts. Announcements will be made via the DDN Management Bulletins or Newsletters as new testing tools and procedures become available.

4.4 Announcement Procedures

Official MIL STD protocols are deposited at the Naval Publications and Forms Center and are announced in the catalogs published by that organization.

Each branch of the military has its own protocol announcement procedure as do non-military government agencies, such as the National Bureau of Standards. Commercial, national, and international standards organizations also have their own review and announcement procedures. It is beyond the scope of this book to summarize each of these procedures. [See *IEEE Communications Magazine*, Vol. 23, No. 1 (Jan. 1985) for an excellent overview of the standardization practices of the various protocol standardization bodies.]

4.4.1 Requests for Comments (RFCs)

Requests for Comments (RFCs) are technical notes describing protocol development and related topics of interest to the ARPANET and DDN research community. Proposed protocols or discussions of protocols while they are under development are found in the RFCs; therefore, this is an important set of documents to monitor if you wish to influence the design of a protocol before it is adopted as a standard, or if you wish to track the progress of DARPA experimental protocols.

The RFCs are maintained online by the NIC on behalf of DARPA and the DDN PMO. Dr. Jonathan B. Postel currently serves as Editor-in-Chief of the RFCs. Researchers wishing to submit an RFC for publication should send it online to POSTEL@USC-ISI.FARPA. The document format for RFCs should follow the standards outlined in the *Instructions for Authors of RFCs*, available online at the NIC in the file RFC:AUTHOR-INSTRUCT.TXT. Section 5.3 explains how to obtain copies of the RFCs and how to be added to the online distribution list for announcements.

4.4.2 DCA Circulars

Information of particular importance to military users is often published as a DCA Circular. DCA Circulars are available in hardcopy from the Defense Technical Information Center (DTIC).

4.4.3 DDN Management Bulletins and Newsletters

The DDN PMO uses online DDN Management Bulletins and informal newsletters to announce the acceptance of new protocols and to inform site personnel, members of the Communications and Operations Group (COG) and implementors, of actions to be taken or decisions made with respect to protocol adoption, change, enhancement, or deletion. The Management Bulletins are distributed by the DDN Network Information Center (NIC) on behalf of the DDN PMO. DDN users wishing to receive the DDN Management Bulletins online may send an electronic message to NIC@SRI-NIC.ARPA or call the NIC telephone "hotline" on (800) 235-3155, and ask to be added to the distribution list. Implementors who are government contractors may also obtain copies of DDN Management Bulletins from their contract monitors.

4.4.4 The TACNEWS Service

TACNEWS is a network service provided by the NIC which permits users to quickly and easily check for announcements, or read the DDN Management Bulletins and Newsletters. It can be accessed from a Terminal Access Controllor (TAC) or from another host.

To access TACNEWS from a TAC, log in and type:

```
@n<Return>
tacnews<Return>
```

To access TACNEWS from another DDN or ARPANET host, make a TELNET connection from that host to the SRI-NIC machine as follows:

```
telnet<Return>
connect SRI-NIC<Return>
```

To access TACNEWS once the TELNET connection is completed, type:

```
@tacnews<Return>
```

SECTION 5. OBTAINING PROTOCOL INFORMATION

5.1 Military Standards

MIL STD protocols can be ordered from:

Naval Publications and Forms Center, Code 3015
5801 Tabor Drive
Philadelphia, PA 19120
Telephone: (215) 697-3321

5.2 The DDN Protocol Handbook

Additional copies of this 1985 DDN Protocol Handbook can be ordered from:

DDN Network Information Center
SRI International, Room EJ291
333 Ravenswood Avenue
Menlo Park, CA 94025
Telephone: (800) 235-3155

The price for the three-volume set is \$110.00, prepaid, to cover the cost of reproduction and handling. Checks should be made payable to SRI International. Copies of the handbook will also be deposited at DTIC.

5.3 Requests for Comments (RFCs)

RFCs are available online or in hardcopy from the NIC. For network users, the online versions can be obtained via FTP from the SRI-NIC host computer (26.0.0.72 on MILNET and 10.0.0.51 on ARPANET) using username "anonymous" and password "guest" and the pathname of RFC:RFCxxx.TXT, where "xxx" equals the number of the RFC desired. An online index is also available with pathname RFC:RFC-INDEX.TXT. Individuals who wish to be added to the RFC notification list should send a message to NIC@SRI-NIC.ARPA requesting that their names be added to the online distribution list. Hardcopies of RFCs may be obtained from the DDN Network Information Center by sending a check or purchase order made payable to SRI International in the amount of \$5.00 for each copy under 100 pages, or \$10.00 for 100 pages and above.

5.4 DDN Management Bulletins and Newsletters

The DDN Management Bulletins and informal DDN Newsletters are available for FTP from the SRI-NIC machine using username "anonymous" and password "guest" and

pathnames of the type DDN-NEWS:DDN-MGT-BULLETIN-xx.TXT and DDN-NEWS:DDN-NEWS-xx.TXT, where "xx" is the number of the bulletin or newsletter desired. All of the newsletters that are still current are online on the NIC machine.

Special quarterly issues of the DDN Newsletter are published both online and in hardcopy. The hardcopy versions are distributed to appropriate military agencies by the DDN PMO. Additional copies are available from the NIC.

Both DDN Management Bulletins and DDN Newsletters can also be read using the TACNEWS service described above.

5.5 NIC Services

The DDN Network Information Center (NIC) assists users in obtaining information pertaining to DoD protocols. The NIC publishes the DDN Protocol Handbook and maintains a NIC Repository of DoD and related protocol documents. It houses the DDN Management Bulletins, the DDN Newsletters, the Requests for Comments (RFC) technical note series, and also produces the *TCP/IP Protocol Implementation and Vendors Guide*. The NIC is a good place to start if you need information.

(800) 235-3155

is the toll-free telephone number to call for user assistance. Service is available Monday through Friday, 7 am to 4 pm, Pacific time.

The NIC host computer is a DEC-2065 running the TOPS-20 operating system with the hostname SRI-NIC and host addresses, 26.0.0.73 (MILNET) and 10.0.0.51 (ARPANET). NIC online services are available 24 hours a day, 7 days a week. Operations personnel are in attendance from 4 am - 11 pm weekdays, and 8 am - 12 pm weekends, Pacific time.

Send online mail to:

NIC@SRI-NIC.ARPA

Send U.S. mail to:

DDN Network Information Center
SRI International, Room RJ291
333 Ravenswood Avenue
Menlo Park, CA 94025

5.6 Other Information Sources

A subscription to the *DoD Index of Specifications and Standards* (DODISS) can be ordered from:

Naval Publications and Printing Service Office
Fourth Naval District
700 Robbins Avenue
Philadelphia, PA 19111

FIPS Standards can be ordered from:

National Technical Information Service (NTIS)
U.S. Dept. of Commerce
5285 Port Royal Road
Springfield, VA 22161
Telephone: (703) 487-4630

ANSI Standards can be ordered from:

American National Standards Institute (ANSI)
Sales Department
1430 Broadway
New York, NY 10018
Telephone: (212) 354-3300

IEEE documents are available from:

Institution of Electrical and Electronic Engineers
445 Hoes Lane
Piscataway, NJ 08854

CCITT documents can be ordered from:

International Telecommunications Union
General Secretariat, Sales Section
Place des Nations
CH-1211 Geneva 20
SWITZERLAND

SECTION 8. DOD MILITARY STANDARD PROTOCOLS

This section contains the official DoD Military Standard Protocols. The X.25 Protocol, Host Front End Protocol, and the Internet Control Message Protocol are also included but are currently under review.

MIL-STD-1777
12 AUGUST 1983

MILITARY STANDARD

INTERNET PROTOCOL



NO DELIVERABLE DATA REQUIRED BY THIS DOCUMENT

WPC/SILK/TCT

MIL-STD-1777
12 August 1983

DEPARTMENT OF DEFENSE
WASHINGTON, D.C. 20301

Internet Protocol

MIL-STD-1777

1. This Military Standard is approved for use by all Departments and Agencies of the Department of Defense.
2. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to: Defense Communications Agency, ATTN: J110, 1860 Wiehle Avenue, Reston, Virginia 22090, by using the self-addressed Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document, or by letter.

MIL-STD-1777
12 August 1983

FOREWORD

This document specifies the Internet Protocol (IP) which supports the interconnection of communication subnetworks. The document includes an introduction to IP with a model of operation, a definition of services provided to users, and a description of the architectural and environmental requirements. The protocol services interfaces and mechanisms are specified using an abstract state machine model.

MIL-STD-1777
12 August 1983

CONTENTS

		<u>Page</u>
Paragraph 1.	SCOPE - - - - -	1
1.1	Purpose - - - - -	1
1.2	Organization - - - - -	1
1.3	Application - - - - -	1
2.	REFERENCED DOCUMENTS- - - - -	2
2.1	Issues of documents - - - - -	2
2.2	Other publications- - - - -	2
3.	DEFINITIONS - - - - -	3
3.1	Definition of terms - - - - -	3
4.	GENERAL REQUIREMENTS- - - - -	7
4.1	Design- - - - -	7
4.2	Internet protocol definition- - - - -	7
4.2.1	Protocol implementation - - - - -	7
4.2.2	Upper layer protocol- - - - -	8
4.2.3	Datagram processing error - - - - -	8
4.2.4	Fragmentation and reassembly mechanisms - - - - -	8
4.2.5	IP evolution- - - - -	9
4.3	Scenario- - - - -	9
4.3.1	Basic model of operation- - - - -	9
5.	SERVICES PROVIDED TO UPPER LAYER- - - - -	11
5.1	Description - - - - -	11
5.2	Datagram service- - - - -	11
5.2.1	Delivery service- - - - -	11
5.3	Generalized network services- - - - -	11
5.3.1	Network parameters- - - - -	11
5.4	Error reporting service - - - - -	12
6.	UPPER LAYER SERVICE/INTERFACE SPECIFICATION - - - - -	13
6.1	Description - - - - -	13
6.2	Interaction primitives- - - - -	13
6.2.1	Service request primitives - - - - -	13
6.2.1.1	SEND- - - - -	13
6.2.2	Service response primitives - - - - -	14
6.2.2.1	DELIVER - - - - -	14
6.3	Extended state machine specification of services provided to upper layer - - - - -	15
6.3.1	Machine instantiation identifier - - - - -	15
6.3.2	State diagram - - - - -	15
6.3.3	State vector - - - - -	15
6.3.4	Data structures - - - - -	15
6.3.4.1	From ULP- - - - -	15
6.3.4.2	To ULP - - - - -	16
6.3.5	Event list - - - - -	16
6.3.6	Events and actions - - - - -	17

MIL-STD-1777
12 August 1983

CONTENTS

		Page
Paragraph 6.3.6.1	EVENT = SEND (from ULP) at time t- - - - -	17
6.3.6.2	EVENT = NULL - - - - -	18
7.	SERVICES REQUIRED FROM LOWER LAYER - - - - -	20
7.1	Description- - - - -	20
7.2	Data transfer- - - - -	20
7.3	Error reporting- - - - -	20
8.	LOWER LAYER SERVICE/INTERFACE SPECIFICATION- - -	21
8.1	Description- - - - -	21
8.2	Interaction primitives - - - - -	21
8.2.1	Service request primitives - - - - -	21
8.2.1.1	SEND - - - - -	21
8.2.2	Service response primitives- - - - -	22
8.2.2.1	SNP_DELIVER- - - - -	22
8.3	Extended state machine specification of services required from lower layer- - - - -	22
8.3.1	Machine instantiation identifier - - - - -	22
8.3.2	State diagram- - - - -	22
8.3.3	State vector - - - - -	22
8.3.4	Data structures- - - - -	22
8.3.4.1	From SNP - - - - -	22
8.3.4.2	To SNP - - - - -	23
8.3.4.3	Dtgm - - - - -	23
8.3.5	Event list - - - - -	24
8.3.6	Events and actions - - - - -	24
8.3.6.1	EVENT = SNP SEND (to SNP)- - - - -	24
8.3.6.2	EVENT = NULL - - - - -	24
9.	IP ENTITY SPECIFICATION- - - - -	25
9.1	Description- - - - -	25
9.2	Overview of IP mechanisms- - - - -	25
9.2.1	Routing mechanism- - - - -	25
9.2.1.1	Internet addresses - - - - -	25
9.2.1.1.1	Internet addressing classes- - - - -	26
9.2.1.1.2	Datascan routing - - - - -	26
9.2.1.2	Routing options- - - - -	27
9.2.1.2.1	Routing types- - - - -	27
9.2.2	Fragmentation and reassembly - - - - -	27
9.2.2.1	Fragment routing - - - - -	28
9.2.2.2	Fragment reassembly- - - - -	28
9.2.2.3	Fragment loss- - - - -	28
9.2.3	Checksum - - - - -	29
9.2.4	Time-to-live - - - - -	29
9.2.5	Type of service- - - - -	29
9.2.6	Data options - - - - -	30
9.2.6.1	Timing information - - - - -	30

MIL-STD-1777
12 August 1983

CONTENTS - Continued

	<u>Page</u>
Paragraph 9.2.7	Error report datagrams- - - - - 31
9.3	Message format for peer exchanges - - - - - 31
9.3.1	Version - - - - - 31
9.3.2	Internet header length- - - - - 31
9.3.3	Type of service - - - - - 32
9.3.4	Total length- - - - - 32
9.3.5	Identification- - - - - 32
9.3.6	Flags - - - - - 33
9.3.7	Fragment offset - - - - - 33
9.3.8	Time-to-live- - - - - 33
9.3.9	Protocol- - - - - 33
9.3.10	Header checksum - - - - - 33
9.3.11	Source address- - - - - 34
9.3.12	Destination address - - - - - 34
9.3.13	Options - - - - - 34
9.3.13.1	Internet options defined- - - - - 35
9.3.14	Padding - - - - - 35
9.3.15	Specific option definitions - - - - - 35
9.3.15.1	End of option list- - - - - 35
9.3.15.2	No operation- - - - - 36
9.3.15.3	Security- - - - - 36
9.3.15.3.1	Security (S field)- - - - - 36
9.3.15.3.2	Compartment (C field)- - - - - 37
9.3.15.3.3	Handling restrictions (H field) - - - - - 37
9.3.15.3.4	Transmission control code (TCC field) - - - - - 37
9.3.15.4	Loose source and record route - - - - - 37
9.3.15.5	Strict source and record route- - - - - 38
9.3.15.6	Record route- - - - - 38
9.3.15.7	Stream identifier - - - - - 38
9.3.15.8	Internet timestamp- - - - - 39
9.4	Extended state machine specification of IP entity - - - - - 39
9.4.1	Machine instantiation identifier- - - - - 39
9.4.2	State diagram - - - - - 39
9.4.3	State vector- - - - - 40
9.4.4	Data structures - - - - - 40
9.4.4.1	State vector- - - - - 41
9.4.4.2	From ULP- - - - - 41
9.4.4.3	To ULP- - - - - 41
9.4.4.4	From SMP- - - - - 42
9.4.4.5	To SMP- - - - - 42
9.4.4.6	Msg- - - - - 42
9.4.5	Event list- - - - - 43
9.4.6	Events and actions- - - - - 43
9.4.6.1	Events and actions decision tables- - - - - 44
9.4.6.1.1	State = inactive, event is SEND from ULP- - - - - 44
9.4.6.1.2	State = inactive, event is SMP_DELIVER from SMP- - - - - 44

MIL-STD-1777
12 August 1983

CONTENTS - Continued

		<u>Page</u>
Paragraph 9.4.6.1.3	State = reassembling, event is SNP_DELIVER	
	from SNP- - - - -	45
9.4.6.1.4	State = inactive, event is TIMEOUT - - - - -	45
9.4.6.2	Decision table functions - - - - -	46
9.4.6.2.1	A_frag - - - - -	46
9.4.6.2.2	Can_frag - - - - -	46
9.4.6.2.3	Checksum_valid - - - - -	47
9.4.6.2.4	Icmp_checksum - - - - -	47
9.4.6.2.5	Need_to_frag - - - - -	48
9.4.6.2.6	Reass_done - - - - -	48
9.4.6.2.7	SNP_params_valid - - - - -	49
9.4.6.2.8	TTL_valid - - - - -	50
9.4.6.2.9	ULP_params_valid - - - - -	51
9.4.6.2.10	Where_dest - - - - -	51
9.4.6.2.11	Where_to - - - - -	52
9.4.6.3	Decision table action procedures - - - - -	52
9.4.6.3.1	Analyze - - - - -	53
9.4.6.3.2	Build&send - - - - -	55
9.4.6.3.3	Compute_checksum - - - - -	56
9.4.6.3.4	Compute_icmp_checksum - - - - -	56
9.4.6.3.5	Error_to_source - - - - -	57
9.4.6.3.6	Error_to_ULP - - - - -	58
9.4.6.3.7	Fragment&send - - - - -	59
9.4.6.3.8	Local_delivery - - - - -	62
9.4.6.3.9	Reassembly - - - - -	63
9.4.6.3.10	Reassembled_delivery - - - - -	65
9.4.6.3.11	Reassembly_timeout - - - - -	66
9.4.6.3.12	Remote_delivery - - - - -	67
9.4.6.3.13	Route - - - - -	68
10.	EXECUTION ENVIRONMENT REQUIREMENTS - - - - -	71
10.1	Description - - - - -	71
10.2	Interprocess communication - - - - -	71
10.3	Timing - - - - -	71

MIL-STD-1777
12 August 1983

CONTENTS - Continued

		<u>Page</u>
FIGURES		
Figure	1. Example host protocol hierarchy - - - - -	7
	2. Example gateway protocol hierarchy - - - - -	8
	3. Base model of operations - - - - -	9
	4. Subnetwork packet - - - - -	10
	5. Internet addresses - - - - -	26
	6. IP header format - - - - -	31
	7. Table of service field - - - - -	32
	8. Control flags field - - - - -	33
	9. Fields of the option-type octet - - - - -	35
	10. Security option format - - - - -	36
	11. A simplified IP state machine - - - - -	40
	12. Transmission order of octets - - - - -	72
	13. Significance of bits - - - - -	72
TABLES		
Table	I. Inactive state decision table when event is SEND from ULP - - - - -	44
	II. Inactive state decision table when event is SNP_DELIVER from SNP - - - - -	44
	III. Reassembling state decision table when event is SNP_DELIVER from SNP - - - - -	45
APPENDICES		
Appendix	A. Data transmission order - - - - -	72

MIL-STD-1777
12 August 1983

1. SCOPE

1.1 Purpose. This standard establishes criteria for the Internet Protocol (IP) which supports the interconnection of communication subnetworks.

1.2 Organization. This standard introduces the Internet Protocol's role and purpose, defines the services provided to users, and specifies the mechanisms needed to support those services. This standard also defines the services required of the lower protocol layer, describes the upper and lower interfaces, and outlines the execution environment services needed for implementation.

1.3 Application. The Internet Protocol (IP) and the Transmission Control Protocol (TCP) are mandatory for use in all DoD packet switching networks which connect or have the potential for utilizing connectivity across network or subnetwork boundaries. Network elements (hosts, front-ends, bus interface units, gateways, etc.) within such networks which are to be used for inter-netting shall implement TCP/IP. The term network as used herein includes Local Area Networks (LANs) but not integrated weapons systems. Use of TCP/IP within LANs is strongly encouraged particularly where a need is perceived for equipment interchangeability or network survivability. Use of TCP/IP in weapons systems is also encouraged where such usage does not diminish network performance.

MIL-STD-1777
12 August 1983

2. REFERENCED DOCUMENTS

2.1 Issues of documents. The following documents of the issue in effect on date of invitation for bids or request for proposal, form a part of this standard to the extent specified herein. (The provisions of this paragraph are under consideration.)

2.2 Other publications. The following documents form a part of this standard to the extent specified herein. Unless otherwise indicated, the issue in effect on date of invitation for bids or request for proposals shall apply. (The provisions of this paragraph are under consideration.)

MIL-STD-1777
12 August 1983

3. DEFINITIONS

3.1 Definition of terms. The definition of terms used in this standard shall comply with FED-STD-1037. Terms and definitions unique to MIL-STD-1777 are contained herein.

- a. Datagram. A self-contained package of data carrying enough information to be routed from source to destination without reliance on earlier exchanges between source or destination and the transporting subnetwork.
- b. Datagram fragment. The result of fragmenting a datagram, also simply referred to as a fragment. A datagram fragment carries a portion of data from the larger original, and a copy of the original datagram header. The header fragmentation fields are adjusted to indicate the fragment's relative position within the original datagram.
- c. Datagram service. A datagram, defined above, delivered in such a way that the receiver can determine the boundaries of the datagram as it was entered by the source. A datagram is delivered with non-zero probability to the desired destination. The sequence in which datagrams are entered into the subnetwork by a source is not necessarily preserved upon delivery at the destination.
- d. Destination. An IP header field containing an internet address indicating where a datagram is to be sent.
- e. DF. Don't Fragment flag: An IP header field that when set "true" prohibits an IP module from fragmenting a datagram to accomplish delivery.
- f. EFTP. Electronic File Transfer Protocol. Electronic mail.
- g. Fragmentation. The process of breaking the data within a datagram into smaller pieces and attaching new internet headers to form smaller datagrams.
- h. Fragment Offset. A field in the IP header marking the relative position of a datagram fragment within the larger original datagram.
- i. FTP. File Transfer Protocol.
- j. Gateway. A device, or pair of devices, which interconnect two or more subnetworks enabling the passage of data from one subnetwork to another. In this architecture, a gateway usually contains an IP module, a Gateway-to-Gateway Protocol (GGP) module, and a subnetwork protocol module (SNP) for each connected subnetwork.

MIL-STD-1777
12 August 1983

- k. Header. Collection of control information transmitted with data between peer entities.
- l. Host. A computer which is a source or destination of messages from the point of view of the communication subnetwork.
- m. ICMP. Internet Control Message Protocol, the collection of error conditions and error message formats exchanged by IP modules in both hosts and gateways.
- n. Identification. An IP header field used in reassembling fragments of a datagram.
- o. IHL. Internet Header Length: an IP header field indicating the number of 32-bit words making up the internet header.
- p. Internet address. A four octet (32 bit) source or destination address composed of a Network field and a REST field. The latter usually contains a local subnetwork address.
- q. Internet datagram. The package exchanged between a pair of IP modules. It is made up of an IP header and a data portion.
- r. Local address. The address of a host within a subnetwork. The actual mapping of an internet address onto local subnetwork addresses is quite general, allowing for many to one mappings.
- s. Local subnetwork. The subnetwork directly attached to host or gateway.
- t. MF. More Fragments flag: an IP header field indicating whether a datagram fragment contains the end of a datagram.
- u. MTU. Maximum Transmission Unit: a subnetwork dependent value which indicates the largest datagram that a subnetwork can handle.
- v. Octet. An eight-bit byte.
- w. Options. The optional set of fields at the end of the IP header used to carry control or routing data. An Options field may contain none, one, or several options, and each option may be one to several octets in length. The options allow ULPs to customize IP's services. The options are also useful in testing situations to carry diagnostic data such as timestamps.
- x. Packet network. A network based on packet-switching technology. Messages are split into small units (packets) to be routed independently on a store and forward basis. This approach pipelines packet transmission to effectively use circuit bandwidth.

MIL-STD-1777
12 August 1983

- y. Padding. An IP header field, one octet in length, inserted after the last option field to ensure that the data portion of a datagram begins on a 32-bit word boundary. The Padding field value is zero.
- z. Protocol. An internet header field used to identify the upper layer protocol that is the source and destination of the data within an IP datagram.
- aa. Reassembly. The process of piecing together datagram fragments to reproduce the original large datagram. Reassembly is based on fragmentation data carried in their IP headers.
- bb. Reliability. One of the service quality parameters provided by the type of service mechanism. The reliability parameter can be set to one of four levels: lowest, lower, higher, or highest. It appears as a two-bit field within the Type of Service field in the IP header.
- cc. Rest. The three-octet field of the internet address usually containing a local address.
- dd. Segment. The unit of data exchanged by TCP modules. This term may also be used to describe the unit of exchange between any transport protocol modules.
- ee. Source. An IP header field containing the internet address of the datagram's point of origin.
- ff. Stream delivery service. The special handling required for a class of volatile periodic traffic typified by voice. The class requires the maximum acceptable delay to be only slightly larger than the minimum propagation time, or requires the allowable variance in packet interarrival time to be small.
- gg. SNP. Subnetwork Protocol: the protocol residing in the subnetwork layer below IP which provides data transfer through the local sub-net. In some systems, an adaptor module must be inserted between IP and the subnetwork protocol to reconcile their dissimilar interfaces.
- hh. TCP. Transmission Control Protocol: a transport protocol providing connection-oriented, end-to-end reliable data transmission in packet-switched computer subnetworks and internetworks.
- ii. TCP segment. The unit of data exchanged between TCP modules (including the TCP header).
- jj. Total Length. An IP header field containing the number of octets in an internet datagram, including both the IP header and the data portion.

MIL-STD-1777
12 August 1983

- kk. Type of Service. An IP header field containing the transmission quality parameters: precedence level, reliability level, speed level, resource trade-off (precedence vs. reliability), and transmission mode (datagram vs. stream). This field is used by the type of service mechanism which allows ULPs to select the quality of transmission for a datagram through the internet.
- ll. UDP. User Datagram Protocol.
- mm. ULP. Upper Layer Protocol: any protocol above IP in the layered protocol hierarchy that uses IP. This term includes transport layer protocols, presentation layer protocols, session layer protocols, and application programs.
- nn. Version. An IP header field indicating the format of the IP header.

MIL-STD-1777
12 August 1983

4. GENERAL REQUIREMENTS

4.1 Design. The Internet Protocol is designed to interconnect packet-switched communication subnetworks to form an internetwork. The IP transmits blocks of data, called internet datagrams, from sources to destinations throughout the internet. Sources and destinations are hosts located on either the same subnetwork or connected subnetworks. The IP is purposely limited in scope to provide the basic functions necessary to deliver a block of data. Each internet datagram is an independent entity unrelated to any other internet datagram. The IP does not create connections or logical circuits and has no mechanisms to promote data reliability, flow control, sequencing, or other services commonly found in virtual circuit protocols.

4.2 Internet protocol definition. This standard specifies a host IP. As defined in the DoD architectural model, the Internet Protocol resides in the internetwork layer. Thus, the IP provides services to transport layer protocols and relies on the services of the lower network layer protocol (See figure 1). In each gateway (a system interconnecting two or more subnets) an IP resides above two or more subnetwork protocol entities. Gateways implement internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) to coordinate routing and other internet control information.

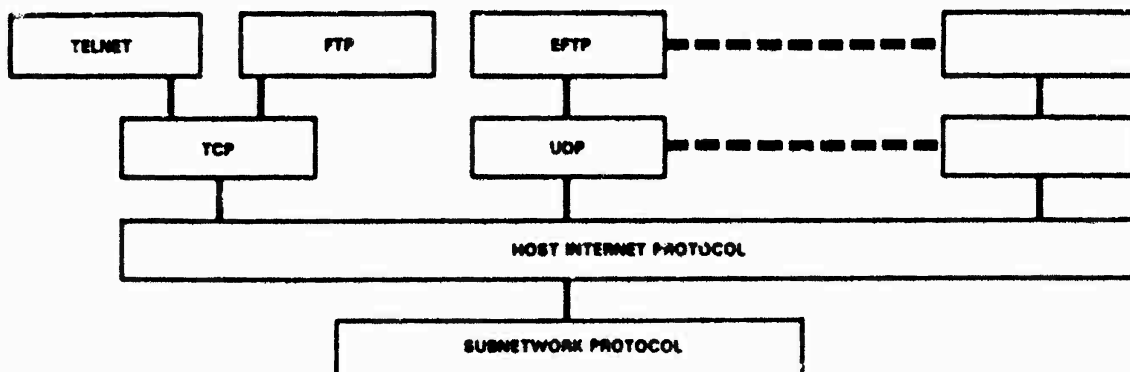


FIGURE 1. Example host protocol hierarchy.

4.2.1 Protocol implementation. In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module (See figure 2).

MIL-STD-1777
12 August 1983

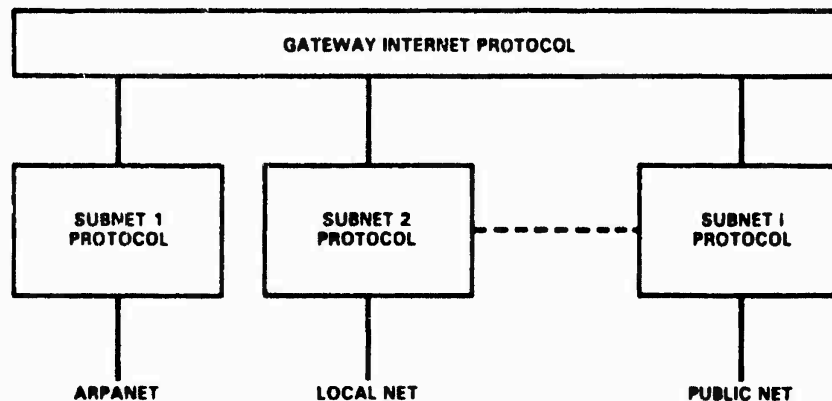


FIGURE 2. Example gateway protocol hierarchy.

4.2.2 Upper layer protocol. A protocol in an upper layer passes data to IP for delivery. IP packages the data as an internet datagram and passes it to the local subnetwork protocol for transmission across the local subnet. If the destination host is on the local subnet, IP sends the datagram through the subnet directly to that host. If the destination host is on a foreign subnet, IP sends the datagram to a local gateway. The gateway, in turn, sends the datagram through the next subnet to the destination host, or to another gateway. Thus, datagrams move from one IP module to another through an interconnected set of subnetworks until they reach their destinations. The sequence of IP modules handling the datagram in transit is called the gateway route. The gateway route is distinct from the lower level node-to-node route supplied by a particular subnetwork. The gateway route is based on the destination internet address. The IP modules share common rules for interpreting internet addresses to perform internet routing.

4.2.3 Datagram processing error. Occasionally, a gateway IP or destination IP will encounter an error during datagram processing. Errors detected may be reported via the Internet Control Message Protocol (ICMP) which is implemented in the internet protocol module.

4.2.4 Fragmentation and reassembly mechanisms. In transit, datagrams may traverse a subnetwork whose maximum packet size is smaller than the size of the datagram. To handle this condition, IP provides fragmentation and reassembly mechanisms. The gateway at the smaller-packet subnet fragments the original datagram into pieces, called datagram fragments, that are small enough for transmission. The IP module in the destination host reassembles the datagram fragments to reproduce the original datagram. IP can support a diverse set of upper layer protocols (ULPs). A transport protocol with real-time requirements, such as the Network Voice Protocol (NVP), can make use of IP's datagram service directly. A transport protocol providing ordered reliable delivery, such as TCP, can build additional mechanisms on top of IP's basic datagram service. Also, IP's delivery service can be customized in some ways to suit the special needs of an upper layer protocol. For example, a predefined gateway route, called a source route, can be supplied for an individual datagram. Each IP module forwards the datagram according to the source route in addition to using the standard routing mechanism.

MIL-STD-1777
12 August 1983

4.2.5 IP evolution. The current Internet Protocol evolved from proposals within the International Federation for Information Processing (IFIP) Technical Committee 6.1, in which internet functions and reliable transport functions were combined in a single protocol. Subsequent development of other ULPs (such as packet speech) led to the separation of these functions to form IP and the Transmission Control Protocol.

4.3 Scenario. The following scenario illustrates the model of operation for transmitting a datagram from one upper layer protocol to another. The scenario is purposely simple so that IP's basic operation is not obscured by the details of interface parameters or header fields.

4.3.1 Basic model of operation. A ULP in host A is to send data to its peer protocol in host B on another subnetwork. In this case, the source and destination hosts are on subnetworks directly connected by a gateway.

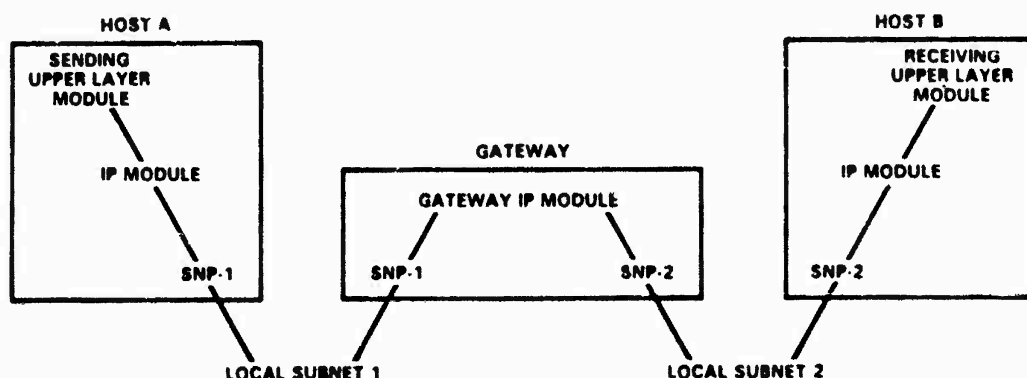


FIGURE 3. Basic model of operation.

- a. The sending ULP passes its data to the IP module, along with the destination internet address and other parameters.
- b. The IP module prepares an IP header and attaches the ULP's data to form an internet datagram. Then, the IP module determines a local subnetwork address from the destination internet address. In this case, it is the address of the gateway to the destination subnetwork. The internet datagram along with the local subnet address is passed to the local subnetwork protocol (SNP).
- c. The SNP creates a local subnetwork header and attaches it to the datagram forming a subnetwork packet. The SNP then transmits the packet across the local subnet.

MIL-STD-1777
12 August 1983

SUBNETWORK PACKET

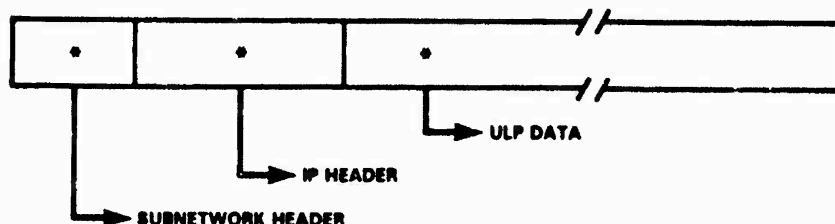


FIGURE 4. Subnetwork packet.

- d. The packet arrives at the gateway connecting the first and second subnetworks. The SNP of the first subnet strips off the local subnetwork header and passes the remainder to the IP module.
- e. The IP module determines from the destination internet address in the IP header that the datagram is intended for a host in the second subnet. The IP module then derives a local subnetwork address for the destination host. That address is passed along with the datagram to the SNP of the second subnetwork for delivery.
- f. The second subnet's SNP builds a local subnetwork header and appends the datagram to form a packet for the second subnetwork. That packet is transmitted across the second subnet to the destination host.
- g. The SNP of the destination host strips off the local subnetwork header and hands the remaining datagram to the IP module.
- h. The IP module determines that the datagram is bound for a ULP within this host. The data portion of the datagram and information from the IP header are passed to the ULP.

Delivery of data across the internet is complete.

MIL-STD-1777
12 August 1983

5. SERVICES PROVIDED TO UPPER LAYER

5.1 Description. This section describes the services offered by the Internet Protocol to upper layer protocols (ULPs). The goals of this section are to provide the motivation for protocol mechanisms and a definition of the functions provided by this protocol. The services provided by IP are: internet datagram service, virtual network service, and error reporting service. A description of each service follows:

5.2 Datagram service. The Internet Protocol shall provide a datagram service between homogeneous upper layer protocols in an internet environment. A datagram service is characterized by data delivery to the destination with non-zero probability; some data may possibly be lost or duplicated. Also, a datagram service does not necessarily preserve the sequence in which data is supplied by the source upon delivery at the destination.

5.2.1 Delivery service. IP shall deliver received data to a destination ULP in the same form as sent by the source ULP. IP shall discard datagrams when insufficient resources are available for processing. IP does not detect datagrams lost or discarded by the subnetwork layer. As part of the delivery service, IP insulates upper layer protocols from subnetwork-specific characteristics. For example, IP maps internet addresses supplied by ULPs into local addresses used by the local subnetwork. Also, IP hides any packet-size restrictions of subnetworks along the transmission path within the internet.

5.3 Generalized network services. IP shall provide to upper layer protocols the ability to select virtual network service parameters. IP shall provide a general command set for the ULPs to indicate the services desired. Thus, the ULPs can tune certain properties of IP and the underlying subnetworks to customize the transmission service according to their needs.

5.3.1 Network parameters. The virtual network parameters fall into two categories: service quality parameters and service options. Service quality parameters influence the transmission service provided by the subnetworks; service options are additional functions provided by IP. A brief description of each follows:

- Service Quality Parameters

- Precedence: attempts preferential treatment for high importance datagrams
- Transmission Mode: datagram vs. stream. Stream mode attempts to minimize delay and delay dispersion through reservation of network resources
- Reliability: attempts to minimize data loss and error rate
- Speed: attempts prompt delivery

MIL-STD-1777
12 August 1983

- Resource Tradeoff: indicates relative importance of speed vs. reliability
- Service Options
 - Security Labelling: identifies datagram for compartmented hosts
 - Source Routing: selects set of gateway IP modules to visit in transit
 - Route Recording: records gateway IP modules encountered in transit
 - Stream Identification: names reserved resources used for stream service
 - Timestamping: records time information
 - Don't Fragment: marks a datagram as an indivisible unit

5.4 Error reporting service. IP shall provide error reports to the upper layer protocols indicating errors detected in providing the above services. In addition, certain errors detected by lower layer protocols or supplied in ICMP messages shall be passed to the ULPs. These reports indicate several classes of errors including invalid arguments, insufficient resources, and transmission errors. The errors that IP must report to ULPs are to be determined for each implementation.

MIL-STD-1777
12 August 1983

6. UPPER LAYER SERVICE/INTERFACE SPECIFICATION

6.1 Description. This section specifies the IP services provided to upper layer protocols and the interface through which these services are accessed. The first part defines the interaction primitives and interface parameters for the upper interface. The second part contains the abstract machine specification of the upper layer services and interaction discipline.

6.2 Interaction primitives. An interaction primitive defines the purpose and content of information exchanged between two protocol layers. Primitives are grouped into two classes based on the direction of information flow. Information passed downward, in this case from a ULP to IP, is called a service request primitive. Information passed upward, in this case from IP to a ULP, is called a service response primitive. Interaction primitives need not occur in pairs. That is, a service request does not necessarily elicit a service "response;" a service "response" may occur independently of a service request. The information associated with an interaction primitive falls into two categories: parameters and data. The parameters describe the data and indicate how the data are to be treated. The data are not examined or modified. The format of the parameters and data are implementation dependent and therefore not specified. A given IP implementation may have slightly different interaction primitives imposed by the execution environment or system design factors. In those cases, the primitives can be modified to include more information or additional primitives can be defined to satisfy system requirements. However, all IPs must provide at least the interaction primitives specified below to guarantee that all IP implementations can support the same protocol hierarchy.

6.2.1 Service request primitives. A single service request primitive supports IP's datagram service, the SEND primitive.

6.2.1.1 SEND. The SEND primitive contains complete control information for each unit of data to be delivered. IP accepts in a SEND at least the following information:

- source address - internet address of ULP sending data
- destination address - internet address of ULP to receive data
- protocol - name of the recipient ULP
- type of service indicators - relative transmission quality associated with unit of data
 - precedence - one of eight levels : (P0, P1, P2, P3, P4, P5, P6, P7) where P0 <= P1 <= P2 <= P3 <= P4 <= P5 <= P6 <= P7
 - reliability - one of two levels : (R0, R1) where R0 <= R1

MIL-STD-1777
12 August 1983

- delay - one of two levels : (D0, D1) where D0 <= D1
- throughput - one of two levels : (T0, T1) where T0 <= T1
- identifier - value optionally provided by this ULP distinguishing this portion of data from others sent by this ULP.
- don't fragment indicator - flag showing whether IP can fragment data to accomplish delivery
- time to live - The value in seconds which indicates the maximum lifetime of data within the internet. Time to live is decremented by one second for each gateway transversed.
- data length - length of data being transmitted
- option data - options requested by a ULP from following list: security, loose or strict source routing, record routing, stream identification, or timestamp (section 9.3.14).
- data - present when data length is greater than zero.

6.2.2 Service response primitives. A single service response primitive supports IP's datagram service, the DELIVER primitive.

6.2.2.1 DELIVER. The DELIVER primitive contains the data passed by a source ULP in a SEND, along with addressing, quality of service, and option information. IP passes in a DELIVER at least the following information:

- source address - internet address of sending ULP
- destination address - internet address of the recipient ULP
- protocol - name of recipient ULP as supplied by the sending ULP
- type of service indicators - relative transmission quality associated with unit of data
 - precedence - one of eight levels : (P0, P1, P2, P3, P4, P5, P6, P7) where P0 <= P1 <= P2 <= P3 <= P4 <= P5 <= P6 <= P7
 - delay - one of two levels : (D0, D1) where D0 <= D1
 - reliability - one of two levels : (R0, R1) where R0 <= R1
 - throughput - one of two levels : (T0, T1) where T0 <= T1
- data length - length of received data (possibly zero)

MIL-STD-1777
12 August 1983

- option data - options requested by source ULP from following list: security, loose source routing, strict source routing, record routing, stream identification, or timestamps (section 6.2.14).
- data - present when data length is greater than zero.

In addition, a DELIVER must contain error reports from IP either together with parameters and data listed above, or independent of that information.

6.3 Extended state machine specification of services provided to upper layer. The extended state machine defines the behavior of the entire service machine from the perspective of the upper layer protocol. An extended state machine definition is composed of a machine instantiation identifier, a state diagram, a state vector, a set of data structures, an event list, and an events and actions correspondence.

6.3.1 Machine instantiation identifier. Each upper interface state machine is uniquely identified by the four interaction primitive parameters: source address, destination address, protocol, and identifier. One state machine instance exists for the SEND and DELIVER primitives whose four parameters carry identical values.

6.3.2 State diagram. The upper interface state machine has a single state which never changes. No diagram is needed.

6.3.3 State vector. The upper interface state machine has a single state which never changes. No state vector is needed.

6.3.4 Data structures. For clarity in the events and actions section, data structures are declared for the interaction primitives and their parameters. A subset of Ada¹ data constructs, common to most high level languages, is used. However, a data structure may be partially typed or completely untyped where specific formats or data types are implementation dependent.

6.3.4.1 From ULP. The from ULP structure holds the interface parameters and data associated with the SEND primitive specified above. This structure directly corresponds to the from ULP structure declared in 9.4.4.2 of the mechanism section. The from ULP structure is declared as:

```
type from_ULP_type is
  record
    source_addr
    destination_addr
    protocol
    type_of_service is
```

¹Ada is a registered trademark of the Department of Defense (Ada Joint Program Office).

MIL-STD-1777
12 August 1983

```

      record
        precedence
        delay
        throughput
        reliability
      end record;
    identifier
    dont_fragment
    time_to_live
    length
    options
    data
  end record;

```

6.3.4.2 To ULP. The to_ULD structure holds interface parameters and data associated with the DELIVER primitive, as specified in section 6.2.2. This structure directly corresponds to the to_ULD structure declared in 9.4.4.3 of the mechanism specification. The to_ULD structure is declared as:

```

type to_ULD_type is
  record
    source_addr
    destination_addr
    protocol
    type_of_service is
      record
        precedence
        delay
        reliability
        throughput
      end record;
    length
    options
    data
    error
  end record;

```

6.3.5 Event list. The events are drawn from the interaction primitives specified in section 6.2 above. An event is composed of a service primitive and an abstract timestamp to indicate the time of event initiation. The event list is as follows:

- a. SEND(from_ULD) at time t
- b. NULL - Although no service request is issued by a ULP, certain conditions within IP or lower layers produce a service response. These conditions can include duplication of data and subnet errors.

MIL-STD-1777
12 August 1983

6.3.6 Events and Actions. The following section defines the set of possible actions elicited by each event.

6.3.6.1 EVENT = SEND (from ULP) at time t.

Actions:

1. DELIVER to ULP at time $t+N$ to the protocol designated by from_ULP.protocol at destination from_ULP.destination_addr with all of the following properties:
 - a. The time elapsed during data transmission satisfies the time-to-live limit, i.e., $N \leq \text{from_ULP.time_to_live}$.
 - b. The quality of data transmission is at least equal to the relative levels specified by from_ULP.type_of_service.
 - c. if (from_ULP.dont_fragment = TRUE) then IP fragmentation has not occurred in transit.
 - d. if (from_ULP.options includes loose source routing) then to_ULP.data has visited in transit at least the gateways named by source route provided by SEND.
 - e. if (from_ULP.options includes strict source routing) then to_ULP.data has visited in transit only the gateways named by source route provided by SEND.
 - f. if (from_ULP.options includes record routing) then the list of nodes visited in transit is delivered in to_ULP.
 - g. if (from_ULP.options includes security labelling) then the security label is delivered in to_ULP.
 - h. if (from_ULP.options includes stream identifier) then the stream identifier is delivered in to_ULP.
 - i. if (from_ULP.options includes internet timestamp) then the internet timestamp is delivered in to_ULP.

OR,

2. DELIVER to the protocol designated by from_ULP.protocol at source from_ULP.source_addr indicating one of the following error conditions:
 - a. destination from_ULP.destination_addr unreachable

MIL-STD-1777
12 August 1983

- b. protocol from_ULP.protocol unreachable
- c. if (from_ULP.dont_fragment = TRUE) then fragmentation needed but prohibited
- d. if (from_ULP.options contains any option) then parameter problem with option.

OR,

3. no action

6.3.6.2 EVENT = NULL.

Actions:

1. DELIVER to the protocol designated by from_ULP.protocol at source from_ULP.source_addr indicating the following error condition:

- a. error conditions in subnet layer

OR,

2. DELIVER to ULP at time t+N to the protocol designated by from_ULP.protocol at destination from_ULP.destination_addr with all of the following properties:
 - a. The time elapsed during data transmission satisfies the time-to-live limit, i.e. $N \leq \text{from_ULP.time_to_live}$.
 - b. The quality of data transmission is at least equal to the relative levels specified by from_ULP.type_of_service.
 - c. if (from_ULP.dont_fragment = TRUE) then IP fragmentation has not occurred in transit.
 - d. if (from_ULP.options includes loose source routing) then to_ULP.data has visited in transit at least the gateways named by source route provided in SEND.
 - e. if (from_ULP.options includes strict source routing) then to_ULP.data has visited in transit only the gateways named by source route provided in SEND.
 - f. if (from_ULP.options includes record routing) then the list of nodes visited in transit is delivered in to_ULP.
 - g. if (from_ULP.options includes security labelling) then the security label is delivered in to_ULP.

MIL-STD-1777
12 August 1983

- h. if (from_ULP.options includes stream identifier)
then the stream identifier is delivered in to_ULP.
- i. if (from_ULP.options includes internet timestamp)
then the internet timestamp is delivered in to_ULP.

MIL-STD-1777
12 August 1983

7. SERVICES REQUIRED FROM LOWER LAYER

7.1 Description. This section describes the minimal services required of the subnetwork layer. The services required are: transparent data transfer between hosts within a subnetwork and error reporting. A description of each service follows.

7.2 Data transfer. The subnetwork layer must provide a transparent data transfer between hosts within a single subnetwork. Only the data to be delivered, and the necessary control and addressing information should be required as input from IP. Intranet routing and subnetwork operation shall be handled by the subnetwork layer itself. The subnetwork need not be a reliable communications medium. Data should arrive with non-zero probability at a destination. Data may not necessarily arrive in the same order as it was supplied to the subnetwork layer, nor is data guaranteed to arrive error free.

7.3 Error reporting. The subnetwork layer shall provide reports to IP indicating errors from the subnetwork and lower layers as feasible. The specific error requirements of the subnetwork layer are dependent on the individual subnetworks.

MIL-STD-1777
12 August 1983

8. LOWER LAYER SERVICE/INTERFACE SPECIFICATION

8.1 Description. This section specifies the minimal subnetwork protocol services required by IP and the interface through which those services are accessed. The first part defines the interaction primitives and their parameters for the lower interface. The second part contains the abstract machine specification of the lower layer services and interaction discipline.

8.2 Interaction primitives. An interaction primitive defines the purpose of information exchanged between two protocol layers. Two kinds of primitives, based on the direction of information flow, are defined. Service requests pass information downward; service responses pass information upward. These primitives need not occur in pairs, nor in a synchronous manner. That is, a request does not necessarily elicit a "response;" a "response" may occur independently of a request. The information associated with an interaction primitive falls into two categories: parameters and data. The parameters describe the data and indicate how the data are to be treated. The data are not examined or modified and the format of interaction primitive information is implementation dependent and is therefore not specified. A given IP implementation may have slightly different interfaces imposed by the nature of the subnetwork or execution environment. Under such circumstances, the primitives can be modified to either include more parameters or have additional primitives defined. However, all IPs must provide at least the interface specified below to guarantee that all IP implementations can support the same protocol hierarchy.

8.2.1 Service request primitives. A single service request primitive is required from the SNP, a SNP_SEND primitive.

8.2.1.1 SEND. The SNP_SEND contains an IP datagram, a destination, and parameters describing the desired transmission quality. The SNP receives in an SNP_SEND at least the following information:

- local destination address - local subnetwork address of destination host or gateway
- type of service indicators - relative transmission quality associated with the datagram
 - precedence - one of eight levels: (P0, P1, P2, P3, P4, P5, P6, P7) where $P0 \leq P1 \leq P2 \leq P3 \leq P4 \leq P5 \leq P6 \leq P7$
 - reliability - one of two levels: (R0, R1) where $R0 \leq R1$
 - delay - one of two levels: (D0, D1) where $D0 \leq D1$
 - throughput - one of two levels: (T0, T1) where $T0 \leq T1$
- length - size of the datagram
- datagram

MIL-STD-1777
12 August 1983

8.2.2 Service response primitives. One service response primitive is required to support IP's datagram service, the SNP_DELIVER primitive.

8.2.2.1 SNP_DELIVER. The SNP_DELIVER contains only a datagram which is an independent entity containing an IP header and data. An IP receives in an SNP_DELIVER at least the following information:

- datagram

In addition, a SNP_DELIVER may contain error reports from the SNP, either together with a datagram or independent of one.

8.3 Extended state machine specification of services required from lower layer. The extended state machine defines the behavior of the entire service machine with respect to the lower layer protocol. An extended state machine definition is composed of a machine instantiation identifier, a state diagram, a state vector, a set of data structures, an event list, and an events and actions correspondence.

8.3.1 Machine instantiation identifier. Each lower interface state machine is uniquely identified by the four values:

- source address
- destination address
- protocol
- identification

These values are drawn from header fields of the datagram passed by the SNP_SEND and SNP_DELIVER primitives. One state machine instance exists for the interaction primitives whose parameters carry the same values.

8.3.2 State diagram. The lower interface state machine has a single state which never changes. No diagram is needed.

8.3.3 State vector. No state vector is needed for the lower interface state machine.

8.3.4 Data structures. For clarity in the events and actions section, data structures are declared for the interaction primitives and their parameters. These structures are declared in a subset of Ada composed of constructs common to most high level languages. However, a data structure may be partially typed or completely untyped where specific formats or data types are implementation dependent.

8.3.4.1 From SNP. The from_SNP structure holds the interface parameters and datagram associated with the SNP_DELIVER primitive, as specified in section 8.2.2.1 This structure directly corresponds to the from_SNP structure declared in section 9.4.4.4 of the mechanism specification. The from_SNP structure is declared as:

MIL-STD-1777
12 August 1983

```

type from_SNP_type is
  record
    source_destination_addr
    dtgm: datagram_type;
    error
  end record;

```

The dtgm element is itself a structure as specified below.

8.3.4.2 To SNP. The to_SNP structure holds the data and parameters associated with the SNP_SEND primitive specified in section 8.3.1. This structure directly corresponds to the to_SNP structure declared in section 9.4.4.5 of the mechanism specification. The to_SNP structure is declared as:

```

type to_SNP_type is
  record
    local_destination_addr
    type_of_service_indicators
    length
    dtgm: datagram_type;
  end record;

```

The dtgm element is itself a structure as specified below.

8.3.4.3 Dtgm. The dtgm structure holds a datagram made up of a header portion and a data portion as specified in section 9.3. A dtgm structure is declared as:

```

type datagram_type is
  record
    version: HALF_OCTET;
    header_length: HALF_OCTET;
    type_of_service: OCTET;
    total_length: TWO_OCTETS;
    identification: TWO_OCTETS;
    dont_frag_flag: BOOLEAN;
    more_frag_flag: BOOLEAN;
    fragment_offset: ONE_N_FIVE_EIGHTHS_OCTETS;
    time_to_live: OCTET;
    protocol: OCTET;
    header_checksum: TWO_OCTETS;
    source_addr: FOUR_OCTETS;
    destination_addr: FOUR_OCTETS;
    options: option_type;
    data: array(1..DATA_LENGTH) of INTEGER;
  end record;

```

```

subtype HALF_OCTET is INTEGER range 0..15;
subtype OCTET is INTEGER range 0..255;
subtype ONE_N_FIVE_EIGHTHS_OCTETS is INTEGER range 0..8191;
subtype TWO_OCTETS is INTEGER range 0..65535;
subtype FOUR_OCTETS is INTEGER range 0..4294967295;

```

MIL-STD-1777
12 August 1983

8.3.5 Event list. The events are drawn from the service primitives specified in section 5.1 above. An event is composed of a service primitive with its parameters and data.

- a. SNP_SEND (to_SNP)
- b. NULL - Although IP issues no service request, certain conditions within the subnet layer elicit a service response.

8.3.6 Events and actions. The following section defines the set of possible actions elicited by each event.

8.3.6.1 EVENT = SNP_SEND (to_SNP).

ACTIONS:

1. SNP_DELIVER Datagram to IP at local destination (LD) with all of the following properties:
 - a. The quality of data transmission is at least equal to the relative levels specified by to_SNP.type_of_service.

OR,

2. no action

8.3.6.2 EVENT = NULL.

ACTIONS:

1. SNP_DELIVER from_SNP indicating the following error condition:
 - a. error conditions within the subnet layer

MIL-STD-1777
12 August 1983

9. IP ENTITY SPECIFICATION

9.1 Description. This section defines the mechanisms of an IP entity supporting the services provided by the IP service machine. The first subsection motivates the specific mechanisms chosen and describes their operation. The second subsection defines the format and use of the IP header fields. The last subsection specifies an extended state machine representation of the protocol entity. The implementation of a protocol entity must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the entity mechanisms, there is always the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret.

9.2 Overview of IP mechanisms. The IP mechanisms are motivated by the IP services, described in section 5 are datagram delivery service, virtual network service, and error reporting service. Each service could be supported by any of a set of mechanisms. The selection of mechanisms is guided by design standards including simplicity, generality, flexibility, and efficiency. The following mechanism descriptions identify the service or services supported, discuss the design criteria used in selection, and explain how the mechanisms work.

9.2.1 Routing mechanism. IP contains an adaptive routing mechanism to support the delivery service. The routing mechanism uses the internet addressing scheme and internet topology data to direct datagrams along the best path between source and destination. The mechanism provides routing options for ULPs needing the flexibility to select routes and record routing information. A distinction is made between names, addresses, and routes. A name indicates the object sought, independent of physical location. An address indicates where the object is and a route indicates how to get there. It is the task of the upper layer protocols to map from names to addresses. The internet protocol maps from internet addresses to local subnet addresses to perform routing through the internet. It is the task of lower layer protocols to route the datagram to the appropriate local subnet destination addresses.

9.2.1.1 Internet addresses. Internet addresses have a fixed length of four octets (32 bits). An internet address begins with a network number followed by a local address (called the REST field). To provide for flexibility in assigning addresses to networks and allow for the large number of small to medium sized networks, there are four formats or classes of internet addresses. These classes are shown in the following diagram:

MIL-STD-1777
12 August 1983

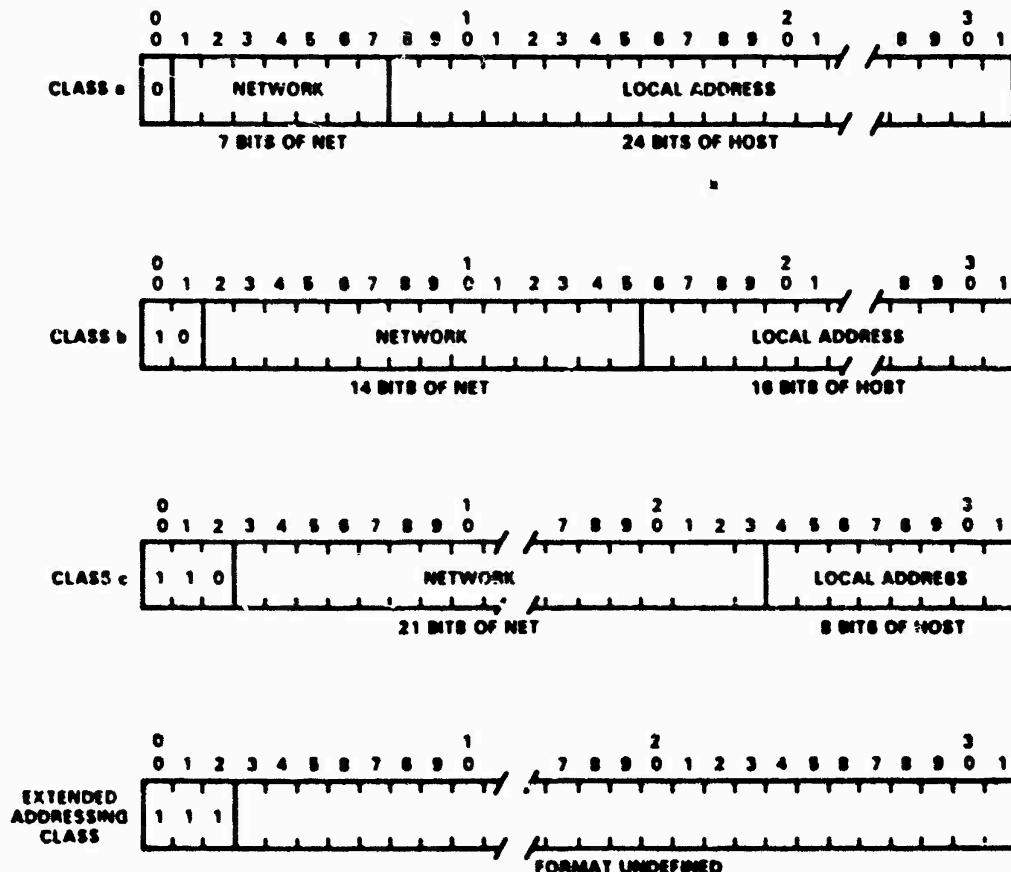


FIGURE 5. Internet addresses.

9.2.1.1.1 Internet addressing classes. In "class a," the high order bit is zero, the next seven bits specify the network, and the last 24 bits specify the local address. In "class b," the high order two bits are one-zero, the next 14 bits are the network and the last 16 bits are the local address. In "class c," the high order three bits are one-one-zero, the next 21 bits are the network and the last eight bits are the local address. In the extended addressing class, the high order three bits are one-one-one, the next 29 bits have no defined format. The mapping between internet addresses and local net addresses should allow a single physical host to act as several distinct internet hosts. Also, some hosts will have several physical interfaces (i.e., be multi-homed). That is, provision must be made for a host to have several physical interfaces to a subnetwork, with each having several logical internet addresses.

9.2.1.1.2 Datascan routing. To route a datagram, an IP module examines the NETWORK field of the internet address indicating the destination for the datagram. If the network number is the same as the IP module's subnetwork, the module uses the REST field of the internet address to derive the local subnet address of the destination host. If the network number does not

MIL-STD-1777
12 August 1983

match, the module determines a local subnet address of a gateway on the best path to the destination subnetwork. In turn, the gateway IP module derives the next local subnet address to either a host or gateway. In this way, the datagram is relayed through the internet to the destination host. In a static environment the routing algorithm is straightforward. However, internet topology tends to change due to hardware or software failure, host availability, or heavy traffic load conditions. Therefore, each host and gateway IP along the gateway route also uses its current knowledge of internet topology to make routing decisions.

9.2.1.2 Routing options. IP provides a mechanism, called source routing, to supplement the gateway's independent routing decisions. This mechanism allows an upper layer protocol to influence the gateway route in which a datagram traverses. The ULP can pass a list of internet addresses, called a source route list, as one of the SEND service request parameters. Each address on the list, except for the last, is an intermediate gateway destination. The last address on the list is the final destination. The source IP module uses its normal routing mechanism to transmit the datagram to the first address in the source route list. Then the gateway IP replaces source route list entry with its own address as known in the environment into which it is forwarding the datagram. Thus, the datagram follows the source route while recording its "inverse" or recorded route.

9.2.1.2.1 Routing types. Two kinds of source routing are provided by IP: loose and strict. With loose source routing, the host and gateway IP modules along the route may use any number of other intermediate gateways to reach the addresses in the source list. With strict source routing, the datagram must travel directly (i.e. through only the directly connected subnetwork indicated by each address) to each address on the source list. When the source route cannot be followed, the source host IP is notified with an error message. For testing or diagnostic purposes, a ULP can acquire a datagram's record route (independent of the source route option) by using the record route mechanism. The sending ULP supplies an empty record route list and indicates that the gateway route is to be recorded in transit. Then, as each gateway IP module on the gateway route relays the datagram, it adds its address as known in the succeeding environment to the record route list. The destination ULP receives the original datagram along with the record route list which, if reversed, provides a source route to the sending ULP. If more gateways are traversed than can be recorded in the list, the additional gateway addresses are not recorded. Problems with the record route option discovered in transit are reported to the source host IP. When using a routing option, the source ULP must provide a large enough route list to accommodate all the routing information expected. The size of a routing option does not change due to adding addresses.

9.2.2 Fragmentation and reassembly. IP contains a fragmentation mechanism for breaking a large datagram into smaller datagrams. This solution to the problems arising from the difference between variable subnetwork capacity provides greater flexibility than legislating a restrictive datagram size that is sufficiently small for any subnetwork on the internet. This mechanism can be overridden using the "don't fragment" option to prevent fragmentation. IP also contains a reassembly mechanism which reverses the fragmentation to

MIL-STD-1777
12 August 1983

enable delivery of intact data portions. Normally, fragmentation is performed only by the IP modules in gateways. There is no need for fragmentation of datagrams within the IP modules of hosts since the amount of data supplied by a source ULP can be limited, thereby avoiding datagrams which are too big to be transmitted through the subnetwork to which the host is attached. When an IP module encounters a datagram that is too big to be transmitted through a subnetwork, it applies its fragmentation mechanism. First, the module divides the data portion of the datagram into two or more pieces. The data must be broken on 8-octet boundaries. For each piece, it then builds a datagram header containing the identification, addressing, and options information needed. Fragmentation data is adjusted in the new headers to correspond to the data's relative position within the original datagram. The result is a set of small datagrams, called fragments, each carrying a portion of the data from the original large datagram. Section 9.4.6.3.7 defines the fragmentation algorithm.

9.2.2.1 Fragment routing. Each fragment is handled independently until the destination IP module is reached. The fragments may follow different gateway routes as internet topology and traffic conditions change. They are also subject to further fragmentation if 'smaller-packet' subnetworks are subsequently traversed. Every IP module must be able to forward a datagram of 68 octets without further fragmentation. This size allows for a header length of up to 60 octets and the minimum data length of 8 octets.

9.2.2.2 Fragment reassembly. To reassemble fragments into the original datagram, an IP module combines all those received having the same value for the identification, source address, destination address, security, and protocol. IP allocates reassembly resources when a "first-to-arrive" fragment is recognized. Based on the fragmentation data in the fragment's header, the fragment is placed in a reassembly area relative to its position in the original datagram. When all the fragments have been received, the IP module passes the data in its original form to the destination ULP. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts send datagrams larger than 576 octets only if they have assurance that the destination is prepared to accept the larger datagrams. The number 576 is selected to allow a reasonable amount of data to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximum internet header size is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of upper layer protocols.

9.2.2.3 Fragment loss. Because the subnetwork may be unreliable, some fragments making up a complete datagram can be lost. IP uses the "time-to-live" data (explained in section 9.2.4 below) to set a timer on the reassembly process. If the timer expires before all the fragments have been collected, IP discards the partially reassembled datagram. Only the destination IP module should perform reassembly. This recommendation is intended to reduce gateway overhead and minimize the chance of deadlock. However, reassembly by private agreement between gateways is transparent to the rest of the

MIL-STD-1777
12 August 1983

internet and is allowed. A ULP can prevent its data from being broken into smaller pieces during transmission. IP provides an override mechanism to prohibit fragmentation called "don't fragment." One example of the "don't fragment" mechanism is the down line loading of a small host containing only a simple boot strap program to accept data from a datagram, storing it in memory, and executing it. Any internet datagram marked "don't fragment" cannot be fragmented by an IP module along the gateway route under any circumstances. If an IP module cannot deliver such a datagram to its destination without fragmenting it, the module discards the datagram and returns an error to the source IP. Note that fragmentation, transmission, and reassembly at the subnetwork layer is transparent to IP and can be used at any time.

9.2.3 Checksum. IP assumes the subnetwork layer to be unreliable regardless of the actual subnetwork protocol present. Therefore, IP provides a checksum mechanism supporting the delivery service to protect the IP header from transmission errors. The data portion is not covered by the IP checksum. If IP enforced a data checksum and discarded datagrams with data checksum failures, it could not support applications that require high throughput and can tolerate a low error rate. An IP module recomputes the checksum each time the IP header is changed. Changes occur in transit during time-to-live reductions, option updates (both explained below), and fragmentation. The checksum is currently a simple one's complement algorithm, and experimental evidence indicates its adequacy. However, the algorithm is provisional and may be replaced by a CRC procedure, depending on future experience.

9.2.4 Time-to-live. As mentioned in the routing discussion above, a datagram's transmission path is subject to changes in internet topology and traffic conditions. Inadvertently, a datagram might be routed on a circuitous path to arrive at its destination after a considerable delay. Or, a datagram could loop through the same IP modules without making real progress towards its destination. Such "old datagrams" reduce internet bandwidth and waste processing time. To prevent these problems, IP provides a mechanism to limit the lifetime of a datagram, called time-to-live. Along with the other sending parameters, a ULP specifies a maximum datagram lifetime in second units. Each IP module on the gateway route decreases the time-to-live value carried in the IP header. If an IP module receives an expired datagram, it discards the datagram. The lifetime limit is in effect until the datagram's data is delivered to the destination ULP. That is, if a datagram is fragmented during transmission, it can still expire during the reassembly process. Section 9.4.4.3 defines the reassembly algorithm use of the time-to-live data.

9.2.5 Type of service. In support of the virtual network service, the type of service mechanism allows upper layer protocols to select the transmission quality. IP passes the type of service (TOS) command set for service quality to the SNP where it is mapped into subnetwork-specific transmission parameters. Not every subnetwork supports all transmission services, but each SNP on the delivery path should make its best effort to match the available subnet services to the desired service quality. The TOS command set includes precedence level, a delay indication, a throughput indication, and a reliability indication. Precedence is a measure of a

MIL-STD-1777
12 August 1983

datagram's importance. A subnetwork may treat high precedence traffic as more important than other traffic by preferentially allocating subnetwork resources especially during time of high load. The eight precedence levels begin with the lowest, Routine, and increase up to the two highest levels, Internetwork Control and Network Control. The highest precedence level, Network Control, is intended for use only within a subnetwork. The Internetwork Control level is intended for use by gateway control originators only. The actual use and access to these precedence levels is the responsibility of each subnetwork. Aside from precedence, the major service choice is a three-way tradeoff between low delay, high reliability, and high throughput. In many networks better performance for one of these parameters is coupled with worse performance for another. Except for very unusual cases, not more than two of these three indications should be set. The use of these service quality indications may increase the cost (in some sense) of the services. Section 9.3.15 specifies the legal values of the type of service indicators to be carried in the datagram header.

9.2.6 Data options. Motivated by the virtual network service, IP provides options to carry certain identification and timing data in a standard manner through the internet. The use of this mechanism by the ULPs is optional, as the name implies, but all options must be supported by each IP implementation. The data options carry three kinds of information: security, stream identification, and timing. The security data is used by DoD hosts needing to transmit security information throughout the internet in a standard manner. The security information (required if classified, restricted, or compartmented traffic is passed) includes security level, compartments, handling restrictions, and transmission control code. The stream identification option provides a way for a stream identifier to be carried both through stream-oriented subnetworks, for example SATNET, and subnets not supporting the stream concept.

9.2.6.1 Timing information. Timing information, in the form of timestamps, is recorded by IP modules as the datagram traverses the internetwork to its destination. The source ULP provides a timestamp list and indicates timing information is to be recorded. The timestamp can be recorded in one of three formats. The first format requires each gateway IP module on the gateway route to register only its timestamp in the next free list entry. The second format requires each gateway IP to register both its internet address and its timestamp. The third format requires a timestamp to be registered only if the next list entry containing a prespecified internet address matches the gateway IP's address. These formats are specified in section 9.2.15. A timestamp is a 32-bit value marking the current time in milliseconds since midnight Universal Time (UT). If the time is not available in milliseconds, or cannot be provided with respect to midnight UT, then any time may be inserted if the high order bit of the timestamp field is set to one, indicating the use of a non-standard value. When using the timestamp option, the source ULP must provide a large enough list to accommodate all the timestamp information expected. The size of the option does not change due to adding timestamps. The initial contents of the timestamp list must be zero or internet address/zero pairs. If the timestamp data area is already full (the pointer exceeds the length) the datagram is forwarded without

MIL-STD-1777
12 August 1983

inserting the timestamp, but the overflow count is incremented by one. If there is some room but not enough for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case, an ICMP parameter problem message may be sent to the source host. Errors encountered by the gateway IPs during timestamp processing are reported to the source IP.

9.2.7 Error report datagrams. The error reporting service motivates a mechanism to generate and process error information. The error mechanism uses the datagram delivery service to transfer the error reports between IP modules.

9.3 Message format for peer exchanges. A summary of the contents of the IP header follows. Note that each tick mark represents a one bit position. Each field description below includes its name, an abbreviation, and the field size. Where applicable, the units, the legal range of values, and a default value appears.

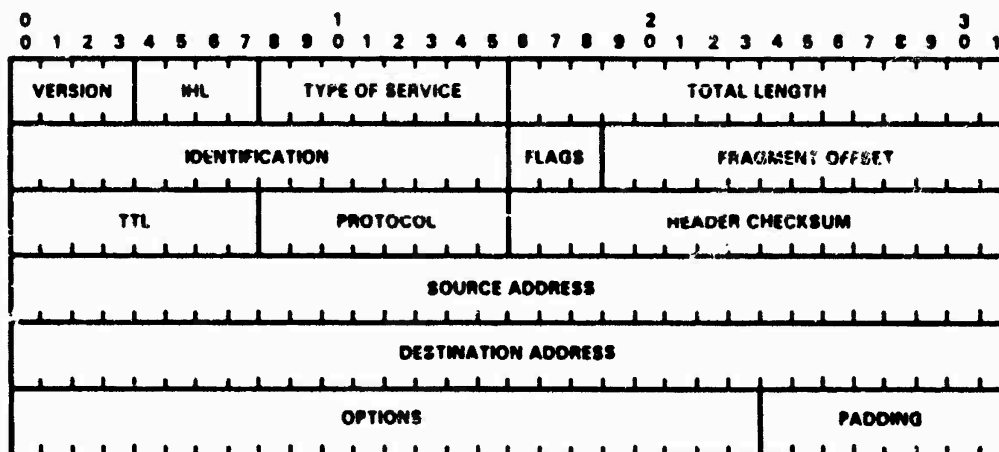


FIGURE 6. IP header format.

9.3.1 Version.

abbrev: VER

field size: 4 bits

The Version field indicates the format of the IP header. This document describes version 4.

9.3.2 Internet header length.

abbrev: IHL

field size: 4 bits

units: 4-octet group

range: 5 - 15

default: 5

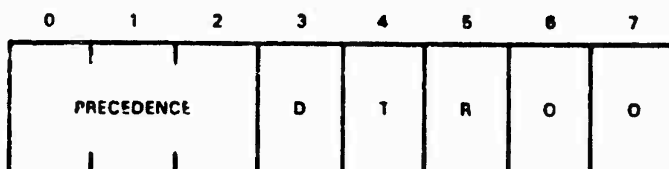
Internet Header Length is the length of the IP header in 32-bit words and points to the beginning of the data. Note that the minimum value for a correct header is 5.

MIL-STD-1777
12 August 1983

9.3.3 Type of service.

abbrev: TOS field size: 8 bits

The Type of Service field contains the IP parameters describing the quality of service desired for this datagram.



Bits 0-2: Precedence
 Bit 3: Delay
 Bits 4: Throughput
 Bits 5: Reliability
 Bit 6-7: Reserved for Future Use.

Precedence	Delay
111 - Network Control	0 - normal
110 - Internetwork Control	1 - low
101 - CRITIC/ECP	
100 - Flash Override	Throughput
011 - Flash	0 - normal
010 - Immediate	1 - high
001 - Priority	
000 - Routine	Reliability
	0 - normal
	1 - high

FIGURE 7. Table of service field.

9.3.4 Total length.

abbrev: TL field size: 16 bits
 units: octets range: 20 - 2**16-1 default: 20

Total Length is the length of the datagram, measured in octets, including header portion and the data portion of the datagram.

9.3.5 Identification.

abbrev: ID field size: 16 bits

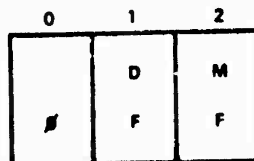
An identifying value used to associate fragments of a datagram. This value is usually supplied by the sending ULP as an interface parameter. If not, IP generates datagram identifications which are unique for each sending ULP.

MIL-STD-1777
12 August 1983

9.3.6 Flags.

abbrev: none field size: 3 bits

This field contains the control flags "don't fragment," which prohibits IP fragmentation and, "more fragments," which helps to identify a fragment's position in the original datagram.



Bit 0: reserved, must be zero
 Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
 Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

FIGURE 8. Control flags field.

9.3.7 Fragment offset.

abbrev: FO field size: 13 bits
 units: 8-octet groups range: 0 - 8191 default: 0

This field indicates the positions of this fragment's data relative to the beginning of the data carried in the original datagram. Both a complete datagram and a first fragment have this field set to zero. Section 9.2.2 describes the fragmentation mechanism.

9.3.8 Time-to-live.

abbrev: TTL field size: 8 bits
 units: seconds range: 0 - 255(=4.25 mins) default: 15

This field indicates the maximum time the datagram is allowed to remain in the internet. If the value of this field drops to zero, the datagram should be destroyed. Section 9.2.4 describes the time-to-live mechanism.

9.3.9 Protocol.

abbrev: PROT field size: 8 bits

This field indicates which ULP is to receive the data portion of the datagram. The numbers assigned to common ULPs are available from the DoD Executive Agent for Protocols.

9.3.10 Header checksum.

abbrev: none field size: 16 bits

MIL-STD-1777
12 August 1983

This field contains the checksum covering the IP header. The checksum mechanism is described in section 9.2.3.

9.3.11 Source address.

abbrev: source field size: 32 bits

This field contains the internet address of the datagram's source host. Internet address formats are discussed in section 9.2.1.

9.3.12 Destination address.

abbrev: dest field size: 32 bits

This field contains the internet address of the datagram's destination host. Internet address formats are discussed in section 9.2.1.

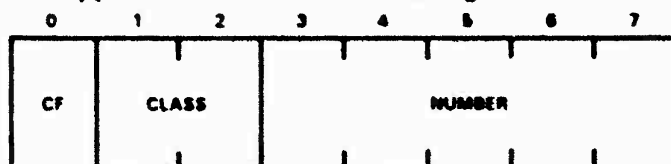
9.3.13 Options.

abbrev: none field size: variable

The option field is variable in length depending on the number and types of options associated with the datagram. The options mechanisms are discussed in sections 9.2.1 and 9.2.6. Options have two formats:

- a. a single octet of option-type, or
- b. a variable length string containing:
 1. an option-type octet,
 2. an option-length octet - counting the option-type octet and option-length octet as well as the option-data octets, and
 3. the actual option-data octets.

The option-type octet is viewed as having 3 fields:



bit 0 - copy flag

0 = not copied, 1 = copied

bits 1-2 - option class

000 = control

001 = reserved for future use

010 = debugging and measurement

011 = reserved for future use

bits 3-7 - option number (defined in the following table)

FIGURE 9. Fields in the option-type octet.

MIL-STD-1777
12 August 1983

9.3.13.1 Internet options defined. The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0000	-	End of Option list: This option occupies only 1 octet; it has no length octet.
0	00001	-	No Operation: This option occupies only 1 octet; it has no length octet.
0	00010	11	Security: Used to carry security level, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DoD requirements.
0	00011	var.	Loose Source Routing: Used to route the datagram based on information supplied by the source.
0	01001	var.	Strict Source Routing: Used to route the datagram based on information supplied by the source.
0	00111	var.	Record Route: Used to trace the route a datagram takes.
0	01000	4	Stream ID: Used to carry the stream identifier.
2	00100	var.	Internet Timestamp: Used to accumulate timing information in transit.

9.3.14 Padding.

abbrev: none field size: variable (8 to 24 bits)

The IP header padding is used to ensure that the IP header ends on a 32-bit boundary. The padding field is set to zero.

9.3.15 Specific option definitions. Each option format is defined below. "Option type" indicates the value of the option-type octet, and "length" indicates the value of the length-octet if appropriate.

9.3.15.1 End of option list.

option type: 0 option length: N/A

This one-octet option marks the end of the option list when it does not coincide with the four-octet boundary indicated by the IP header length. This field is used following the last option, not the end of each option, and need only be used if the last option would not otherwise coincide with the end of the IP header. This option may be introduced or deleted upon fragmentation as needed.

MIL-STD-1777
12 August 1983

9.3.15.2 No operation.

option type: 1 option length: N/A

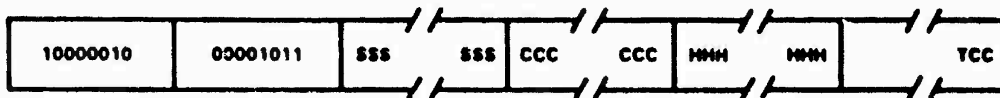
This option may be used between options, for example, to align the beginning of a subsequent option on a 32-bit boundary. This option may be introduced or deleted upon fragmentation as needed.

9.3.15.3 Security.

option type: 130 option length: 11

This option (required if classified, restricted, or compartmented traffic is passed) provides a way for hosts to send Security level, Compartmentation, Handling Restriction Codes and User Groups (TCC) parameters through subnetworks in a standard manner. This option must be copied on fragmentation. This option appears at most once in a datagram.

The format for this option is as follows:



TYPE = 130 LENGTH = 11

FIGURE 10. Security Option Format.

9.3.15.3.1 Security (S field).

length: 16 bits

This field specifies one of 16 levels of security, eight of which are reserved for future use.

00000000	00000000	- Unclassified
11110001	00110101	- Confidential
01111000	10011010	- EPTO
10111100	01001101	- MMM
01011110	00100110	- PROC
10101111	00010011	- Restricted
11010111	10001000	- Secret
01101011	11000101	- Top Secret
00110101	11100010	- (Reserved for future use)
10011010	11110001	- (Reserved for future use)
01001101	01111000	- (Reserved for future use)
00100100	10111101	- (Reserved for future use)
00010011	01011110	- (Reserved for future use)
10001001	10101111	- (Reserved for future use)

MIL-STD-1777
12 August 1983

11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)

9.3.15.3.2 Compartments (C field).

length = 16 bits

This field contains an all zero value when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency (DIA).

9.3.15.3.3 Handling restrictions (H field).

length = 16 bits

The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 60-19, "Standard Security Markings."

9.3.15.3.4 Transmission control code (TCC field).

length = 24 bits

This field provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs and are available from Headquarters, DCA (Code 530).

9.3.15.4 Loose source and record route.

option type: 131 option length: variable

The loose source route option provides a way for the source ULP of a datagram to supply routing information to be used by IP modules along the gateway route. At the same time, the "inverse" route is recorded in the option field. This option is not copied on fragmentation. It appears at most once in a datagram. The option begins with the option type code. The second octet is the option length which includes the option type octet, the length octet, the pointer octet, and the source route list. The third octet is a pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and its smallest legal value is 4. A loose source route list is composed of one or more internet addresses identifying intermediate gateways to be visited in transit. Each internet address is 4 octets long. When a gateway in the source route list is visited, the gateway address (as known in the environment into which the datagram is being forwarded) replaces that list entry. The size of this option is fixed by the source. It cannot change to accommodate additional information. The routing options are described in section 9.2.1.1.

MIL-STD-1777
12 August 1983

9.3.15.5 Strict source and record route.

option type: 137 option length: variable

The strict source route option provides a way for the source ULP of a datagram to name the exact set of IP modules to be visited along the gateway route. At the same time, the "inverse" route is recorded in the option field. This option must be copied on fragmentation. It appears at most once in a datagram. The option begins with the option type code. The second octet is the option length which includes the option type octet, the length octet, the pointer octet, and the source route list. The third octet is a pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and its smallest legal value is 4. A strict source route list is composed of one or more internet addresses identifying the gateways to be visited in transit. The datagram must visit exactly the gateways listed, traversing only the directly connected subnetworks indicated in the route list addresses. When a gateway in the source route list is visited, the gateway address (as known in the environment into which the datagram is being forwarded) replaces that list entry. The size of this option is fixed by the source. It cannot change to accommodate additional information. Routing options are described in section 9.2.1.1.

9.3.15.6 Record route.

option type: 7 option length: variable

The record route option provides a way to record a datagram's gateway route. This option is not copied on fragmentation. It appears at most once in a datagram. The option begins with the option type code. The second octet is the option length which includes the option type code, the length octet, and the return route list. The third octet is a pointer into the route data indicating the octet which begins the next area to store a route address. The pointer is relative to this option, and the smallest legal value for the pointer is 4. A record route list is composed of a series of internet addresses. Each internet address is 4 octets long. The source ULP provides a route list with zero value entries. As each gateway is visited in transit, it registers its address in the next free entry (indicated by the pointer). When the pointer is greater than the length, the record route list is full. No additional addresses are recorded, even if more are visited before arriving at the destination. The size of this option is fixed by the source. It cannot change to accommodate additional information. The routing options are described in section 9.2.1.1.

9.3.15.7 Stream identifier.

option type: 136 option length: 4

This option provides a way for 16-bit stream identifiers to be carried through the internet for use by subnetworks supporting the stream concept such as the SATNET. The stream identifier appears in the third and fourth octets of

MIL-STD-1777
12 August 1983

the option. This option must be copied on fragmentation. It appears at most once in a datagram.

9.3.15.8 Internet timestamp.

option type: 68 option length: variable

This option allows timing information to be gathered as a datagram travels through the internet to its destination. This option is not copied upon fragmentation and so appears only in the first fragment. This option may appear at most once in a datagram. The first octet is the option type. The second octet is the length of the option including the option type octet, the length octet, the pointer octet, the overflow/flag octet, and each timestamp or address/timestamp pair. The third octet is a pointer into the timestamp list identifying the octet beginning the space for the next timestamp. The pointer is relative to the beginning of this option; its smallest legal value is 5. The fourth octet is shared by overflow and format flag information. The first four bits record the number of IP modules that could not register timestamps due to lack of space. The second four bits indicate the format of the timestamp list:

- 0 - timestamps only, stored in consecutive 32-bit words
- 1 - each timestamp is preceded with the internet address of the registering entity
- 2 - reserved for future use
- 3 - the internet address fields are prespecified by the source ULP. An IP module only registers its timestamp if its address matches the next one in the list.

The size of this option is fixed by the source. It cannot change to accommodate additional information. The internet timestamp option is described in section 9.2.6.

9.4 Extended state machine specification of IP entity. The IP entity is specified with an extended state machine made up of a set of states, a set of transitions between states, and a set of input events causing the state transitions. The following specification is made up of a machine instantiation identifier, a state diagram, a state vector, data structures, an event list, and a correspondence between events and actions. In addition, an extended state machine has an initial state whose values are assumed at state machine instantiation.

9.4.1 Machine instantiation identifier. Each datagram is an independent unit. Therefore, one state machine instance exists for each datagram. Each state machine is uniquely named by the four values, source address, destination address, protocol, and identification. These values are drawn from parameters of the interaction primitives specified in sections 6.2 and 8.2.

9.4.2 State diagram. The following diagram depicts a simplified IP state machine.

MIL-STD-1777
12 August 1983

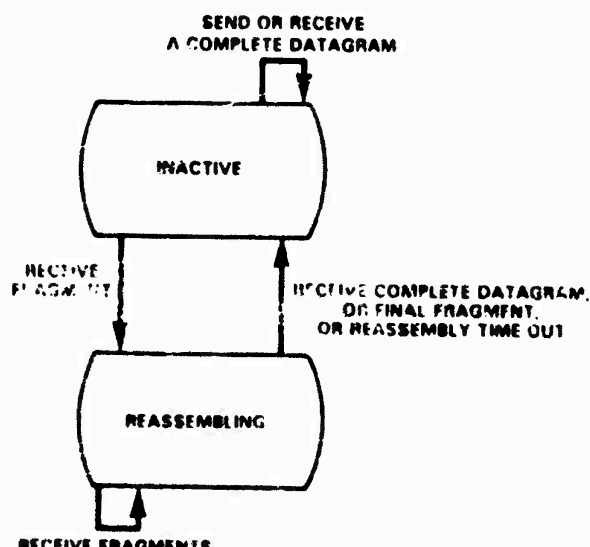


FIGURE 11. A simplified IP state machine.

9.4.3 State vector. A state vector consists of the following elements:

- STATE NAME = (inactive, reassembling)
- REASSEMBLY RESOURCES = control information and storage needed to reassemble fragments into the original datagram, including:
 - a. reassembly map: a representation of each 8-octet unit of data and its relative location within the original datagram.
 - b. timer: value of the reassembly timer in unit seconds from 0 to 255.
 - c. total data length: size of the data carried in datagram being reassembled.
 - d. header: storage area for the header portion of the datagram being reassembled.
 - e. data: storage area for the data portion of the datagram being reassembled.

A state machine's initial state is INACTIVE with unused reassembly resources.

9.4.4 Data structures. The IP state machine references certain data areas corresponding to the state vector, and each interaction primitive: SEND, DELIVER, SNP_SEND and SNP_DELIVER. For clarity in the events and actions section, data structures are declared in Ada for these data areas. However,

MIL-STD-1777
12 August 1983

a data structure may be partially typed or completely untyped where specific formats or data types are implementation dependent.

9.4.4.1 State vector. The definition of an IP state vector appears in section 9.4.3 above. A state_vector structure is declared as:

```
state_vector: state_vector_type;

type state_vector_type is
  record
    state_name: (INACTIVE, REASSEMBLING);
    reassembly_map
    timer
    total_data_length
    header
    data
  end record;
```

9.4.4.2 From ULP. The from ULP structure holds the interface parameters and data associated with the SEND primitive, as specified in section 6.2.1. The from_ULP structure is declared as:

```
from_ULP: from_ULP_type;

type from_ULP_type is
  record
    source_addr
    destination_addr
    protocol
    type_of_service is
      record
        precedence
        delay
        throughput
        reliability
      end record;
    identifier time_to_live
    don't_fragment
    length
    data
    options
  end record;
```

9.4.4.3 To ULP. The to ULP structure holds interface parameters and data associated with the DELIVER primitive, as specified in section 6.2.2. The to_ULP structure is declared as:

```
to_ULP : to_ULP_type;

type to_ULP_type is
  record
    source_addr
```

MIL-STD-1777
12 August 1983

```
destination_addr
protocol
type_of_service is
  record
    precedence
    delay
    throughput
    reliability
  end record;
length
data
options
error
end record;
```

9.4.4.4 From SNP. The from_SNP structure holds the interface parameters and datagram associated with the SNP_DELIVER primitive, as specified in section 8.3.2. The from_SNP structure is declared as:

```
type from_SNP_type is
  record
    local_destination_addr
    dtgm: datagram_type;
    error
  end record;
```

The dtgm element is itself a structure as specified below.

9.4.4.5 To SNP. The to_SNP structure holds the data and parameters associated with the SNP_SEND primitive specified in section 8.3.1. The to_SNP structure is declared as:

```
type to_SNP_type is
  record
    local_destination_addr
    type_of_service_indicators
    length
    dtgm: datagram_type;
  end record;
```

The dtgm element is itself a structure as specified below.

9.4.4.6 Dtgm. A dtgm structure holds a datagram made up of a header portion and a data portion as specified in section 9.3. A dtgm structure is declared as:

```
type datagram_type is
  record
    version: HALF_OCTET;
    header_length: HALF_OCTET;
    type_of_service: OCTET;
    total_length: TWO_OCTETS;
```

MIL-STD-1777
12 August 1983

```
identification: TWO_OCTETS;  
dont_frag_flag: BOOLEAN;  
more_frag_flag: BOOLEAN;  
fragment_offset: ONE_N_FIVE_EIGHTHS_OCTETS;  
time_to_live: OCTET;  
protocol: OCTET;  
header_checksum: TWO_OCTETS;  
source_addr: FOUR_OCTETS;  
destination_addr: FOUR_OCTETS;  
options: OPTION_TYPE;  
data: array(1..DATA_LENGTH) of INTEGER;  
end record;
```

```
subrecord HALF_OCTET is INTEGER range 0..15;  
subrecord OCTET is INTEGER range 0..255;  
subrecord ONE_N_FIVE_EIGHTHS_OCTETS is INTEGER range 0..8191;  
subrecord TWO_OCTETS is INTEGER range 0..65535;  
subrecord FOUR_OCTETS is INTEGER range 0..4294967296;  
subrecord OPTION_TYPE is zero or more of the following:  
    security;  
    loose source routing;  
    strict source routing;  
    record route;  
    stream identifier;  
    internet timestamp;
```

9.4.5 Event list. The event list is made up of the interaction primitives specified in sections 6.2 and 8.2 and the services provided by the execution environment defined in section 10. The following list defines the set of possible events in an IP state machine:

- a. SEND from ULP — A ULP passes interface parameters and data to IP for delivery across the internet (see section 6.2.1.1).
- b. SNP_DELIVER from SNP — SNP passes to IP a datagram received from subnetwork protocol (see section 8.2.2.1).
- c. TIMEOUT — The timing mechanism provided by the execution environment indicates a previously specified time interval has elapsed (see section 10.3).

9.4.6 Events and actions. This section is organized in three parts. The first part contains a decision table representation of state machine events and actions. The decision tables are organized by state; each table corresponds to one event. The second part specifies the decision functions appearing at the top of each column of a decision table. These functions examine attributes of the event and the state vector to return a set of decision results. The results become the elements of each column. The third part specifies action procedures appearing at the right of every row. Each row of the decision table combines the decision results to determine appropriate event processing. These procedures specify event processing algorithms in detail.

MIL-STD-1777
12 August 1983

9.4.6.1 Events and actions decision tables.

9.4.6.1.1 State = inactive, event is SEND from ULP.

TABLE I. Inactive state decision table when event is SEND from ULP.

Actions:

ULP PARAMS VALID?	WHERE DEST ?	NEED TO FRAG?	CAN FRAG ?	
NO	d	d	d	ERROR TO ULP (PARAM__PROBLEM)
YES	ULP	d	d	LOCAL DELIVERY
YES	REMOTE	NO	d	BUILD & SEND
YES	REMOTE	YES	NO	ERROR TO ULP (CAN'T__FRAG)
YES	REMOTE	YES	YES	FRAGMENT & SEND

Comments:

A ULP passes data to IP for internet delivery. IP validates the interface parameters, determines the destination, and dispatches the ULP data to its destination.

Legend

d = "don't care" condition

9.4.6.1.2 State = inactive, event is SNP DELIVER from SNP.

TABLE II. Inactive state decision table when event is SNP DELIVER from SNP.

Actions:

CHECK- SUM VALID?	SNP PARAMS VALID?	TTL VALID ?	WHERE TO ?	A FRAG ?	ICMP CHECK- SUM?	
NO	d	d	d	d	d	DISCARD
YES	NO	d	d	d	d	ERROR TO SOURCE (PARAM__PROBLEM)
YES	YES	NO	d	d	d	ERROR TO SOURCE (EXPIRED__TTL)
YES	YES	YES	ULP	NO	d	REMOTE DELIVERY
YES	YES	YES	ULP	YES	d	REASSEMBLE: STATE: = REASSEMBLING
YES	YES	YES	ICMP	NO	NO	DISCARD
YES	YES	YES	ICMP	NO	YES	ANALYZE
YES	YES	YES	ICMP	YES	d	REASSEMBLE: STATE: = REASSEMBLING
YES	YES	YES	REMOTE	d	d	ERROR TO SOURCE (HOST__UNREACH)

MIL-STD-1777
12 August 1983

Comments:

The SNP has delivered a datagram from another IP. IP validates the datagram header, and either delivers the data from a complete datagram to its destination within the host or begins reassembly for a datagram fragment.

Legend

d = "don't care" condition

9.4.6.1.3 State = reassembling, event is SNP DELIVER from SNP.

TABLE III. Reassembling state decision table when event is SNP DELIVER from SNP.

Actions:

CHECK-SUM VALID?	SNP PARAMS VALID?	TTL VALID ?	WHERE TO ?	A FRAG ?	REASS DONE ?	ICMP CHECK-SUM?	
NO	d	d	d	d	d	d	DISCARD
YES	NO	d	d	d	d	d	ERROR_TO_SOURCE (PARAM_PROBLEM)
YES	YES	NO	d	d	d	d	ERROR_TO_SOURCE (EXPIRED_TTL)
YES	YES	YES	ULP	NO	d	d	REMOTE_DELIVERY; STATE: = INACTIVE
YES	YES	YES	ULP	YES	NO	d	REASSEMBLE
YES	YES	YES	ULP	YES	YES	d	REASS_DELIVERY; STATE: = INACTIVE
YES	YES	YES	ICMP	NO	d	NO	DISCARD
YES	YES	YES	ICMP	NO	d	YES	ANALYZE; STATE: = INACTIVE
YES	YES	YES	ICMP	YES	NO	d	REASSEMBLE
YES	YES	YES	REMOTE	d	d	d	ERROR_TO_SOURCE (HOST_UNREACH)

Comments:

The SNP has delivered a datagram associated with an earlier received datagram fragment. IP validates the header and either continues the reassembly process with the datagram fragment or delivers the data from the completed datagram to its destination within the host.

Legend

d = "don't care" condition

9.4.6.1.4 State = inactive, event is TIMEOUT.

Actions: reassembly timeout; state: = INACTIVE

Comment:

The time-to-live period of the datagram being reassembled has elapsed. The incomplete datagram is discarded; the source IP is informed.

MIL-STD-1777
12 August 1983

9.4.6.2 Decision table functions. The following functions examine information contained in interface parameters, interface data, and the state vector to make decisions. These decisions can be thought of as further refinements of the event and/or state. The return values of the functions represent decisions made. The decision functions appear in alphabetical order.

9.4.6.2.1 A frag? The a_frag function examines certain fields in an incoming datagram's header to determine whether the datagram is a fragment of a larger datagram. The data effects of this algorithm are:

a. Data examined only:

from_SNP.dtgm.fragment_offset
from_SNP.dtgm.more_frag_flag

b. Return values:

NO - the datagram has not been fragmented
YES - the datagram is a part of a larger datagram

if ((from_SNP.dtgm.fragment_offset = 0) --contains the
 beginning
 and (from_SNP.dtgm.more_frag_flag = 0)) --and the end
 of the data

then return NO --therefore it is an unfragmented datagram

else return YES; --otherwise it contains only a portion of
 the data
 --and is a fragment.

end if;

9.4.6.2.2 Can frag? The can_frag function examines the "don't fragment" flag of the interface parameters allowing fragmentation. The data effects of this function are:

a. Data examined only:

from_ULP.dont_fragment

b. Return values:

NO - "don't fragment" flag is set, preventing fragmentation
YES - "don't fragment" flag is NOT set, to allow fragmentation

if (from_ULP.dont_fragment = TRUE)
then return NO
else return YES
end if;

MIL-STD-1777
12 August 1983

9.4.6.2.3 Checksum valid? The checksum_valid function examines an incoming datagram's header to determine whether it is free from transmission errors. The data effects of this function are:

a. Data examined only:

- from_SNP.dtgm.version
- from_SNP.dtgm.header_length
- from_SNP.dtgm.type_of_service
- from_SNP.dtgm.total_length
- from_SNP.dtgm.identification
- from_SNP.dtgm.dont_frag_flag
- from_SNP.dtgm.more_frag_flag
- from_SNP.dtgm.fragment_offset
- from_SNP.dtgm.time_to_live
- from_SNP.dtgm.protocol
- from_SNP.dtgm.source_addr
- from_SNP.dtgm.destination_addr
- from_SNP.dtgm.options

b. Return values:

NO -- checksum did not check, indicating header fields
contain errors
YES -- checksum was consistent

--The checksum algorithm is the 16-bit one's complement
--of the one's complement sum of all 16-bit words in
--the IP header. For purposes of computing the checksum,
--the checksum field is set to zero.

--implementation dependent action

9.4.6.2.4 Icmp checksum? The icmp_checksum function computes the checksum of the ICMP control message carried in the data portion of the incoming datagram. The data effects of this procedure are:

a. Data examined:

- from_SNP.dtgm.data

b. Return values:

NO -- checksum did not check indicating the control
message contains errors
YES -- checksum was consistent

--The checksum algorithm is the 16-bit one's complement of
--the one's complement sum of all 16-bit words
--in ICMP control message. For purposes of computing the
checksum,

MIL-STD-1777
12 August 1983

--the checksum field (octets 2-3) is set to zero.

--implementation dependent action

9.4.6.2.5 Need to frag? The need_to_frag function examines the interface parameters and data from a ULP to determine whether the data can be transmitted as a single datagram or must be transmitted as two or more datagram fragments. The data effects of this function are:

a. Data examined only:

from ULP.length
from ULP.options

b. Return values:

NO - one datagram is small enough for the subnetwork
YES - datagram fragments are needed to carry the data

--Compute the datagram's length based on the length of data,
--the length of options, and the standard datagram header size.

```
if (( from ULP.length + (number of bytes of option data)
      + 20) > maximum transmission unit of the local
      subnetwork)
then return YES
else return NO;
end if;
```

9.4.6.2.6 Reass done? The reass_done function examines the incoming datagram and the reassembly resources to determine whether the final fragment has arrived to complete the datagram being reassembled. The data effects of this function are:

a. Data examined only:

state_vector.reassembly_map
state_vector.total_data_length
from SNP.dtgm.total_length
from SNP.dtgm.more_frag_flag
from SNP.dtgm.header_length

b. Return values:

NO - more fragments are needed to complete reassembly
YES - this is the only fragment needed to complete
reassembly

--The total data length of the original datagram, as computed
--from "tail" fragment, must be known before completion is
--possible.

MIL-STD-1777
12 August 1983

```
if (state_vector.total_data_length = 0)
then
  --Check incoming datagram for "tail."

  if (from_SNP.dtgm.more_frag_flag = FALSE)
  then
    --Compute total data length and see if data in
    --this fragment fill out reassembly map.

    if (state_vector.reassembly_map from 0 to
        (((from_SNP.dtgm.total_length -      -- total data
          (from_SNP.dtgm.header_length*4)+7)/8)-- length
        +7)/8 is set)
    then return YES;
    end if;

  else
    --Reassembly cannot be complete if total data length unknown.
    return NO;
  end if;

else --Total data length is already known. See if data
  --in this fragment fill out reassembly map.

  if ( all reassembly map from 0 to
      (state_vector.total_data_length+7)/8 is set)
  then return YES; -- final fragment
  else return NO;  -- more to come
  end if;
end if;
```

9.4.6.2.7 SNP params valid? The SNP_params_valid function examines the interface parameters and the datagram received from the local subnetwork protocol to see if all values are within legal ranges and no errors have occurred. The data effects of this function are:

a. Data examined only:

```
from_SNP.dtgm.version
from_SNP.dtgm.header_length
from_SNP.dtgm.total_length
from_SNP.dtgm.protocol
other information/errors from SNP
```

b. Return values:

```
NO - some value or values are illegal or an error has
    occurred
YES - examined values are within legal ranges and no
     errors have occurred
```

MIL-STD-1777
12 August 1983

```

if ( --The current IP header version number is 4.
    (from_SNP.dtgm.version /= 4)

    --The minimal IP header is 5 32-bit units in length.
or (from_SNP.dtgm.header_length < 5)

    --The smallest legal datagram contains only a header and is
    --20 octets in length.
or (from_SNP.dtgm.total_length < 20)

    --The legal protocol identifiers are
    --available from the DoD Executive Agent for Protocols.
or (from_SNP.dtgm.protocol is not one of the acceptable identi-
    fiers)

)

then return NO

else if (any implementation dependent values received from the
    SNP are illegal or indicate error conditions)
    then return NO
    else return YES;    --Otherwise, all values look good.
    end if;
end if;

```

9.4.6.2.8 TTL valid? The TTL_valid function examines the IP header time-to-live field of an incoming datagram to determine whether the datagram has exceeded its allowed lifetime. The data effects of this function are:

a. Data examined only:

from_SNP.dtgm.time_to_live

b. Return values:

NO - the datagram has expired

YES - the datagram has some life left in it

```

--Decrement from_SNP.dtgm.time_to_live field by the maximum
--of either the amount of time elapsed since the last IP module
--handled this datagram (if known) or one second.

```

```

if ( from_SNP.dtgm.time_to_live
    - maximum (number of seconds elapsed since last IP, 1)
    <= 0 )
then return NO
else return YES;

```

MIL-STD-1777
12 August 1983

9.4.6.2.9 ULP params valid? The ULP_params_valid function examines the interface parameters received from a ULP to see if all values are within legal ranges and desired options are supported. The data effects of this function are:

a. Data examined only:

from ULP.time_to_live
from ULP.options

b. Return values:

NO - some value is illegal or a desired option is not supported.

YES - examined values are within legal ranges and desired options can be supported.

if (

--The time-to-live value must be greater than zero to
--allow IP to transmit it at least once.

(from ULP.time_to_live < 0)

or --The options requested are inconsistent.

--implementation dependent action

or --Other implementation dependent values are invalid.

--implementation dependent action)

then return NO

else return YES;

end if;

9.4.6.2.10 Where dest? The where_dest function determines the destination of an outgoing datagram by examining the destination address supplied by the ULP. The data effects of this function are:

a. Data examined only:

from ULP.destination_addr

b. Return values:

ULP - destination is an upper layer protocol at this location

REMOTE - destination is some remote location

--Examine the destination address field of the datagram header.

MIL-STD-1777
12 August 1983

```

if (from_SNP.dtg.destination_addr /= this site's address)
then return REMOTE
else return ULP;
end if;

```

9.4.6.2.11 Where to? The where to function determines the destination of the incoming datagram by examining the address fields and options fields of the datagram header. The data effects of this function are:

a. Data examined only:

```

from_SNP.dtg.destination_addr
from_SNP.dtg.protocol
from_SNP.dtg.options

```

b. Return values:

```

ULP - destination is an upper layer protocol at this
      location
ICMP - destination is this IP module because the
      datagram carries an ICMP control message
REMOTE - destination is some remote location

```

--The source route influences the datagram's gateway route.

```

if ((from_SNP.dtg.options contains the source routing
    option) and (all source route list addresses have
    not been visited))
then return REMOTE;
end if;

```

--Examine the destination address field of the datagram header.

```

if (from_SNP.dtg.destination_addr /= this site's address)
then
  --It's destined for another site.
  return REMOTE
else
  --It's destined for this site.
  if (from_SNP.dtg.protocol = the ICMP protocol identifier)
  then return ICMP
  else return ULP;
  end if;
end if;

```

9.4.6.3 Decision table action procedures. The following action procedures represent the set of actions an IP state machine should perform in response to

MIL-STD-1777
12 August 1983

a particular event and internal state. These procedures have been organized and designed for clarity and are intended as guidelines. Although implementors may in fact reorganize for better performance, the data effects of the resulting implementations must not differ from those specified below.

9.4.6.3.1 Analyze. The analyze procedure examines datagrams containing ICMP control messages from other IP modules. In general, error handling is implementation dependent. However, guidelines are provided to identify classes of errors and suggest appropriate actions. The data effects of this procedure are:

a. Data examined:

from_SNP.dtgm.protocol
from_SNP.dtgm.data

b. Data modified:

implementation dependent

For simplicity, it is assumed that the data area can be accessed as a byte array.

--Examine the first octet in the data portion to identify the
--error type and subsequent format.

begin

case from_SNP.dtgm[1] of

when 3 => --Destination Unreachable Message

--The errors in the "unreachable" class
--should be passed to the ULP indicating data delivery
--to the destination is unlikely if not impossible.
--The second octet identifies what level was unreachable.

case from_SNP.dtgm[2] of

when 0 => --net unreachable

when 1 => --host unreachable

when 2 => --protocol unreachable

when 3 => --port unreachable

when 4 => --fragmentation needed and don't fragment set

MIL-STD-1777
12 August 1983

```
        when 5 => --source route failed

    end case;

when 11 => --Time Exceeded Message

    --The "time-out" errors are usually not passed
    --to the ULP but should be recorded for network
    --monitoring uses.

    case from_SNP.dtg of

        when 0 => --Time to live exceeded in transit

            when 1 => --Fragment reassembly time exceeded

        end case;

    when 12 => --Parameter Problem Message

        --This error is generated by a gateway IP to indicate
        --a problem in the options field of a datagram header.
        --Octet 5 contains a pointer which identifies
        --the octet of the original header containing the error.

    when 4 => --Source Quench Message

        --This message indicates that a datagram has been
        --discarded for congestion control. The ULP should
        --be informed so that traffic can be reduced.

    when 5 => --Redirect Message

        --This message should result in a routing table update
        --by the IP module. Octets 5-8 contain the new value
        --for the routing table. It is not passed to the ULP.

    when 8 => --Echo Datagram

    when 0 => --Echo Reply Datagram

    when 13 => --Timestamp Datagram

    when 14 => --Timestamp Reply Datagram

    when 15 => --Information Request Message

    when 16 => --Information Reply Message

end case;
```

MIL-STD-1777
12 August 1983

--Call the route procedure to determine a local destination
--from the internet destination address supplied by the ULP.

route;

--Request the execution environment to pass the contents of to_SNP
--to the local subnetwork protocol for transmission.

TRANSFER to_SNP to the SNP.

NOTE: The format of the from_ULP elements is unspecified, allowing an implementor to assign data types for the interface parameters. If those data types differ from the IP header types, the assignment statements above become type conversions.

9.4.6.3.3 Compute checksum. The compute checksum procedure calculates a checksum value for a datagram header so that transmission errors can be detected by a destination IP. The data effects of this procedure are:

a. Data examined:

to_SNP.dtgm.version
to_SNP.dtgm.header_length
to_SNP.dtgm.type_of_service
to_SNP.dtgm.total_length
to_SNP.dtgm.identification
to_SNP.dtgm.dont_frag_flag
to_SNP.dtgm.more_frag_flag
to_SNP.dtgm.fragment_offset
to_SNP.dtgm.time_to_live
to_SNP.dtgm.protocol
to_SNP.dtgm.source_addr
to_SNP.dtgm.destination_addr
to_SNP.dtgm.options

b. Data modified:

to_SNP.dtgm.header_checksum

--The checksum algorithm is the 16-bit one's complement of
--the one's complement sum of all 16-bit words
--in the IP header. For purposes of computing the checksum,
--the checksum field is set to zero.

--implementation dependent action

9.4.6.3.4 Compute icmp checksum. The compute icmp checksum procedure computes the checksum of the ICMP control message carried in the data portion of an outgoing datagram. The data effects of this procedure are:

MIL-STD-1777
12 August 1983

9.4.6.3.2 Build&send. The build&send procedure builds an outbound datagram in the to_SNP structure from the interface parameters and data in from_ULP and passes it to the SNP for transmission across the subnet. The data effects of this procedure are:

a. Data examined:

from_ULP.source_addr	from_ULP.time_to_live
from_ULP.destination_addr	from_ULP.dont_fragment
from_ULP.protocol	from_ULP.options
from_ULP.type_of_service	from_ULP.length
from_ULP.identifier	from_ULP.data

b. Data modified:

to_SNP.dtgm	to_SNP.type_of_service_indicators
to_SNP.length	to_SNP.local_destination_addr

--Fill in each IP header field with information from from_ULP or --standard values.

```

to_SNP.dtgm.version: = 4;      --Current IP version is 4.
to_SNP.dtgm.type_of_service_indicators: = from_ULP.type_of_service;
to_SNP.dtgm.identification: = from_ULP.identifier; --If ID is not given
--by ULP, the IP must
--supply its own.

to_SNP.dtgm.dont_frag_flag: = from_ULP.dont_fragment;
to_SNP.dtgm.more_frag_flag: = false;
to_SNP.dtgm.fragment_offset: = false;
to_SNP.dtgm.time_to_live: = from_ULP.time_to_live;
to_SNP.dtgm.protocol: = from_ULP.protocol;
to_SNP.dtgm.source_addr: = from_ULP.source_addr;
to_SNP.dtgm.destination_addr: = from_ULP.destination_addr;
to_SNP.dtgm.options: = from_ULP.options;
to_SNP.dtgm.header_length: = 5 + (number of bytes of option data)/4;
to_SNP.dtgm.total_length: = (to_SNP.dtgm.header_length)*4
                           + (from_ULP.length);

```

--Call compute_checksum to compute and set the checksum.

```
compute_checksum;
```

--And, fill in the data portion of the datagram.

```

to_SNP.dtgm.data[0..from_ULP.length-1]: = from_ULP.data[0..
                                         from_ULP.length-1];

```

--Set the type of service and length fields for the SNP.

```

to_SNP.type_of_service_indicators: = to_SNP.dtgm.type_of_service;
to_SNP.length: = to_SNP.dtgm.total_length;

```

MIL-STD-1777
12 August 1983

a. Data examined:

to_SNP.dtgm.data

b. Data modified:

to_SNP.dtgm.data[2-3]

--The checksum algorithm is the 16-bit one's complement of
--the one's complement sum of all 16-bit words
--in ICMP control message. For purposes of computing the
checksum,
--the checksum field (octets 2-3) is set to zero.

--implementation dependent action

9.4.6.3.5 Error to source. The error_to_source procedure formats and returns an error report to the source of an erroneous or expired datagram. The data effects of this procedure are:

a. Parameters:

error_param : (PARAM_PROBLEM, EXPIRED_TTL,
PROTOCOL_UNREACH);

b. Data examined:

from_SNP.dtgm

c. Data modified:

to_SNP.dtgm to_SNP.local_destination_addr
to_SNP.length to_SNP.type_of_service_indicators

--Format and transmit an error datagram to the source IP.

to_SNP.dtgm.version : = 4; --standard IP version
to_SNP.dtgm.header_length : = 5; --standard header size
to_SNP.dtgm.type_of_service : = 0; --routine service quality
to_SNP.dtgm.identification : = select new value;
to_SNP.dtgm.more_frag_flag : = FALSE;
to_SNP.dtgm.dont_frag_flag : = FALSE;
to_SNP.dtgm.fragment_offset : = 0;
to_SNP.dtgm.time_to_live : = 60; --or value large enough to
 --allow delivery
to_SNP.dtgm.protocol : = this number will be assigned by
 DoD Executive Agent for Protocols;
to_SNP.dtgm.source_addr : = from_SNP.dtgm.destination_addr;
to_SNP.dtgm.destination_addr : = from_SNP.dtgm.source_addr;

MIL-STD-1777
12 August 1983

--The data section carries the ICMP control message.
--The first octet identifies the message type, the remaining
--octets carry related information.

case error_param of

where PARAM_PROBLEM =>
to_SNP.dtgm.data[0]: = 12; --ICMP type = Parameter Problem
to_SNP.dtgm.data[1]: = 0; --Code = problem with option
to_SNP.dtgm.data[4]: = position of error octet;

where EXPIRED_TTL =>
to_SNP.dtgm.data[0]: = 11; --ICMP type = Time Exceeded
to_SNP.dtgm.data[1]: = 0; --Code = TTL exceed in transit

where PROTOCOL_UNREACH =>
to_SNP.dtgm.data[0]: = 3; --ICMP type = Dest. Unreachable
to_SNP.dtgm.data[1]: = 2; --Code = protocol unreachable

end case;

--The bad datagram's header plus the first 64 bytes of its
--data section (a total of "N" octets) is copied in following
--the ICMP information.

to_SNP.dtgm.data[8..N+3]: = from_SNP.dtgm.data[0..N-1];
to_SNP.dtgm.total_length: = from_SNP.header_length*4 + N + 8;
compute_icmp_checksum;

--Compute checksum, determine the route for the error datagram,
--the type of service indicators, and the datagram size for the SNP.

compute_checksum;
to_SNP.type_of_service_indicators: = 0;
to_SNP.length := to_SNP.dtgm.total_length;
route;

--Request the execution environment to pass the contents of to_SNP
--to the local subnet protocol for transmission.

TRANSFER to_SNP to the SNP.

9.4.6.3.6 Error to ULP. The error_to_ULD procedure returns an error report to a ULP which has passed invalid parameters or has requested a service that cannot be provided. The data effects of this procedure are:

a. Parameters:

error_param: (PARAM_PROBLEM, CAN'T_FRAGMENT,
NET_UNREACH, PROTOCOL_UNREACH,
PORT_UNREACH);

MIL-STD-1777
12 August 1983

b. Data examined:

implementation dependent

c. Data modified:

to ULP.error
implementation dependent parameters

- The format of error reports to a ULP is implementation
- dependent. However, included in the report should be
- a value indicating the type of error, and some information
- to identify the associated data or datagram.

to ULP.error := error_param;
--implementation dependent action

9.4.6.3.7 Fragment&send. The fragment&send procedure breaks data that is too big to be transmitted through the subnetwork as a single datagram into smaller pieces for transmission in several datagrams. Normally, hosts do not send datagrams too big to go through their own network. The data effects of the procedure are:

a. Data examined only:

from <u>ULP.source_addr</u>	from <u>ULP.length</u>
from <u>ULP.destination_addr</u>	from <u>ULP.data</u>
from <u>ULP.protocol</u>	from <u>ULP.options</u>
from <u>ULP.identifier</u>	from <u>ULP.time_to_live</u>
from <u>ULP.dont_fragment</u>	

b. Data modified:

to <u>SNP.dtg</u>	to <u>SNP.type_of_service_indicators</u>
to <u>SNP.length</u>	

c. Local variables:

number_of_fragments -- number of small datagrams created
from user data

data_per_fragment -- the number of octets in each small
datagram

number_frag_blocks -- the number of 8-octet blocks in each
small datagram

data_in_last_frag -- the number of octets in the last
datagram

j -- loop counter for each fragment generated

--Compute the fragmentation variables.

--The amount of data per fragment equals the max datagram size
less

MIL-STD-1777
12 August 1983

```
--the length of the datagram header.
data_per_fragment: = maximum subnet transmission unit
                    - (20 + number of bytes of option data);

number_frag_blocks: = data_per_fragment/8;

number_of_fragments: = (from ULP.length + (data_per_fragment-1)) / data_per_fragment;

data_in_last_frag: = from_ULP.length modulo data_per_fragment;

--Create the first fragment and transmit it to the SNP.

to_SNP.dtgm.version: = 4;
to_SNP.dtgm.header_length: = 5 + (number bytes of option
                                data/4);
to_SNP.dtgm.total_length: = to_SNP.dtgm.header_length
                            + data_per_fragment;
to_SNP.dtgm.identification: = from_ULP.identifier;
to_SNP.dtgm.dont_frag_flag: = from_ULP.dont_fragment;
                                --this will be false
to_SNP.dtgm.more_frag_flag: = TRUE;
to_SNP.dtgm.fragment_offset: = 0;

to_SNP.dtgm.time_to_live: = from_ULP.time_to_live;
to_SNP.dtgm.protocol: = from_ULP.protocol;
to_SNP.dtgm.source_addr: = from_ULP.source_addr;
to_SNP.dtgm.destination_addr: = from_ULP.destination_addr;
to_SNP.dtgm.options: = from_ULP.options;
to_SNP.dtgm.data[0..data_per_fragment-1]: =
                                from_ULP.data[0..data_per_
                                fragment-1];

--Set the datagram's header checksum field.
compute_checksum;

--Call route to determine the subnetwork address of the
--destination.
route;

--Also set the length and type of service indicators.
to_SNP.length := to_SNP.dtgm.total_length;
to_SNP.type_of_service_indicators := to_SNP.dtgm.type_
of_service;

--Request the execution environment to pass the first fragment
--to the SNP.
TRANSFER to_SNP to the local subnetwork protocol.

--Format and transmit successive fragments.

for j in 1..number_of_fragments-1 loop
```

MIL-STD-1777
12 August 1983

```
--The header fields remain the same as in the first
--fragment, EXCEPT for:

    if ("copy" flag present in any options) --most signi-
                                              --ficant bit
                                              --of option
                                              --octet

    then --put ONLY "copy" options into options fields and
        --adjust length fields accordingly.

        to_SNP.dtg.h.options: = (options with "copy" flag);
        to_SNP.dtg.h.header_length: = 5 +
                                      (number of copy options
                                       octets/4);

    else --only standard datagram header present

        to_SNP.dtg.h.header_length: = 5;

    end if;

--Append data and set fragmentation fields.
if (j /= number_of_fragments-1)

    then --middle fragment(s)

        to_SNP.dtg.h.more_frag_flag: = TRUE;
        to_SNP.dtg.h.fragment_offset: = j*number_frag_blocks;
        to_SNP.dtg.h.total_length: = to_SNP.dtg.h.header_length
                                      + data_per_fragment;
        to_SNP.dtg.data[0..data_per_fragment-1]: =
            from_ULP.data[j*data_per_frag-
            ment.. (j*data_per_fragment
            + data_per_fragment-1)];

    else --last fragment

        to_SNP.dtg.h.more_frag_flag: = FALSE;
        to_SNP.dtg.h.fragment_offset: = j*number_frag_blocks;
        to_SNP.dtg.h.total_length: = to_SNP.dtg.h.header_length*4
                                      + data_in_last_frag;
        to_SNP.dtg.data[0..data_in_last_frag-1]: =
            from_ULP[j*data_per_fragment..
            (j*data_per_fragment+ data_in_
            last_frag-1)];

    end if;

--Call checksum to set the datagram's header checksum field.
checksum;
```

MIL-STD-1777
12 August 1983

```
--Call route to determine the subnetwork address of the
--destination.
route;

--Also set the length and type of service indicators.
to_SNP.length := to_SNP.dtgm.total_length;
to_SNP.type_of_service_indicators := to_SNP.dtgm.type_of_
service;

--Request the execution environment to pass this fragment
--to the SNP.
TRANSFER to_SNP to the local subnetwork protocol.

end loop;
```

A fragmentation algorithm may vary according to implementation concerns but every algorithm must meet the following requirements:

- a. A datagram must not be fragmented if dtgm.dont_frag_flag is true.
- b. The amount of data in each fragment (except the last) must be broken on 8-octet boundaries.
- c. The first fragment must contain all options carried by the original datagram, except padding and no-op octets.
- d. The security, source routing, and stream identification options (i.e. marked with "copy" flag, MSB in option octet) must be carried by all fragments, if present in the original datagram.
- e. The first fragment must have to_SNP.dtgm.fragment_offset set to zero.
- f. All fragments, except the last, must have to_SNP.dtgm.more_frag_flag set true.
- g. The last fragment must have the to_SNP.dtgm.more_frag_flag set false.

9.4.6.3.8 Local delivery. The local_delivery procedure moves the interface parameters and data in the from_ULP structure to the to_ULP structure and delivers it to an in-host ULP. The data effects of this procedure are:

- a. Data examined:

from_ULP.destination_addr	from_ULP.length
from_ULP.source_addr	from_ULP.data
from_ULP.protocol	from_ULP.options
from_ULP.type_of_service	

MIL-STD-1777
12 August 1983

b. Data modified:

to_ULP.source_addr	to_ULP.length
to_ULP.destination_addr	to_ULP.data
to_ULP.protocol	to_ULP.options
to_ULP.type_of_service	

--Move the interface parameters and data from the input
--structure, from ULP, directly to the output structure,
--to_ULP, for delivery to a local ULP.

```

from_ULP.destination_addr: = to_ULP.destination_addr;
from_ULP.source_addr      : = to_ULP.source_addr;
from_ULP.protocol         : = to_ULP.protocol;
from_ULP.type_of_service : = to_ULP.type_of_service;
from_ULP.length           : = to_ULP.length;
from_ULP.data             : = to_ULP.data;
from_ULP.options          : = to_ULP.options;

```

--Request the execution environment to pass the contents of
--to_SNP to the local subnet protocol for transmission.

TRANSFER to_ULP to to_ULP.protocol.

9.4.6.3.9 Reassembly. The reassembly procedure reconstructs an original datagram from datagram fragments. The data effects of this procedure are:

a. Data examined:

from_SNP.dtg

b. Data modified:

```

state_vector.reassembly_map
state_vector.timer
state_vector.total_data_length
state_vector.header
state_vector.data

```

c. Local variables:

j -- loop counter

data_in_frag -- the number of octets of data in received fragment

MIL-STD-1777
12 August 1983

```

    data_in_frag: = (from_SNP.dtgm.total_length-from_SNP.
        dtgm.header_length*4);

--Put data in its relative position in the data area of the
state vector.

    state_vector.data[from_SNP.dtgm.fragment_offset*8..
        from_SNP.dtgm.fragment_offset*8+data_in_frag]: =
        from_SNP.dtgm.data[0..data_in_frag-1];

--Fill in the corresponding entries of the reassembly map
--representing each 8-octet unit of received data.

    for j in (from_SNP.dtgm.fragment_offset)..
        ((from_SNP.dtgm.fragment_offset + data_in_frag +
            7)/8) loop

        state_vector.reassembly_map[j]: = 1;
    end loop;

--Compute the total datagram length from the "tail-end"
--fragment.

    if (from_SNP.dtgm.more_frag_flag = FALSE)
    then state_vector.total_data_length: =
        from_SNP.dtgm.fragment_offset*8 +
        data_in_frag;

    end if;

--Record the header of the "head-end" fragment.

    if (from_SNP.dtgm.fragment_offset = 0)
    then state_vector.header := from_SNP.dtgm;
    end if;

--Reset the reassembly timer if its current value is less
--than the time-to-live field of the received datagram.

    state_vector.timer: = maximum
        (from_SNP.dtgm.time_to_live, state_vector.timer);

```

A reassembly algorithm may vary according to implementation concerns, but each one must meet these requirements:

- a. Every destination IP module must have the capacity to receive a datagram 576 octets in length, either in one piece or in fragments to be reassembled.

MIL-STD-1777
12 August 1983

- b. The header of the fragment with `from_SNP.dtgm.fragment_offset` equal to zero (i.e. the "head-end" fragment) becomes the header of the reassembling datagram.
- c. The total length of the reassembling datagram is calculated from the fragment with `from_SNP.dtgm.more_frag_flag` equal to zero (i.e., the "tail-end" fragment).
- d. A reassembly timer is associated with each datagram being reassembled. The current recommendation for the initial timer setting is 15 seconds. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium. That is, data rate multiplied by timer value equals reassembly capacity (e.g. 10Kb/s X 15secs = 150Kb).
- e. As each fragment arrives, the reassembly timer is reset to the maximum of `state_vector.reassembly_resources.timer` and `from_SNP.dtgm.time_to_live` in the incoming fragment.
- f. The first fragment of the datagram being reassembled must contain all options, except padding and no-op octets.
- g. The `source_addr`, `destination_addr`, protocol, and identifier of the first fragment received must be recorded. All subsequent fragments' `source_addr`, `destination_addr`, protocol, and identifier will be compared against those recorded. Those fragments which do not match will be discarded.
- h. As each fragment arrives, the security and precedence fields, if available, must be checked. If the security level of the fragment does not match the security level of the datagram or if the precedence level of the fragment does not match the precedence level of the datagram, the datagram being assembled is discarded. Also, an error datagram is returned to the source IP to report the "mismatched security/precedence" error.
- i. If the reassembly timer expires, the datagram being reassembled is discarded. Also, an error datagram is returned to the source IP to report the "time exceeded during reassembly" error.

9.4.6.3.10 Reassembled delivery. The `reassembled_delivery` procedure decomposes the datagram that has been reassembled in the state vector into interface parameters and data, then delivers them to a ULP. The data effects of this procedure are:

a. Data examined:

```
state_vector.header.destination_addr
state_vector.header.source_addr
state_vector.header.protocol
state_vector.header.type_of_service
```

MIL-STD-1777
12 August 1983

```
state_vector.header.header_length
state_vector.header.total_length
state_vector.header.options
state_vector.data
```

b. Data modified:

```
to_ULP.destination_addr    to_ULP.length
to_ULP.source_addr         to_ULP.data
to_ULP.protocol            to_ULP.options
to_ULP.type_of_service
```

```
to_ULP.destination_addr: = state_vector.header.destina-
                           tion_addr;
to_ULP.source_addr      : = state_vector.header.source_addr;
to_ULP.protocol         : = state_vector.header.protocol;
to_ULP.type_of_service : = state_vector.header.type_of_
                           service;
to_ULP.length           : = state_vector.header.total_
                           length;
                           - state_vector.header.header_
                           length*4;
to_ULP.options          : = state_vector.header.options;
to_ULP.data              : = state_vector.data;
```

9.4.6.3.11 Reassembly timeout. The reassembly timeout procedure generates an error datagram to the source IP informing it of the datagram's expiration during reassembly. The data effects of the procedure are:

a. Data examined:

```
state_vector.header
state_vector.data
```

b. Data modified:

```
to_SNP.dtgm      to_SNP.type_of_service_indicators
to_SNP.length    to_SNP.header_length
```

--Format and transmit an error datagram to the source IP.

```
to_SNP.dtgm.version      : = 4; --standard IP version
to_SNP.dtgm.header_length : = 5; --standard header size
to_SNP.dtgm.type_of_service: = 0; --routine service quality

to_SNP.dtgm.identification : = new value selected
to_SNP.dtgm.more_frag_flag : = FALSE;
to_SNP.dtgm.dont_frag_flag : = FALSE;
```

MIL-STD-1777
12 August 1983

```

to_SNP.dtgm.fragment_offset : = 0;
to_SNP.dtgm.time_to_live     : = 60;
to_SNP.dtgm.protocol         : = this number will be as-
                                signed by the DoD Executive
                                Agent for Protocols;
to_SNP.dtgm.source_addr      : = state_vector.header.desti-
                                nation_addr;
to_SNP.dtgm.destination_addr: = state_vector.header.source_
                                addr;

```

--If the fragment received is the first fragment, then the
--data section carries the ICMP error message, the header of the
--timed-out datagram, and its first 54 bytes of data. If frag-
--ment zero is not available then no time exceeded need be sent
--at all.

```

to_SNP.dtgm.data[0]: = 12; --ICMP type = Time Exceeded
to_SNP.dtgm.data[1]: = 1;  --Code = fragment reassembly
                           timeout

```

--Copy in the timed-out datagram's header plus the first
--64 bytes of its data section (assumed to be of length "N").

```

to_SNP.dtgm.data[8..N+3] := state_vector[0..N-1];
to_SNP.dtgm.total_length := to_SNP.header_length*4 + N + 8;
compute_icmp_checksum;

```

--Compute datagram's header checksum, determine the route for
--the datagram, the type of service indicators, and the
--datagram size for the SNP.

```

compute_checksum;
to_SNP.type_of_service_indicators := 0;
to_SNP.length := to_SNP.dtgm.total_length;
route;

```

--Request the execution environment to pass the contents of
--to_SNP to the local subnet protocol for transmission.

TRANSFER to_SNP to the SNP.

9.4.6.3.12 Remote delivery. The remote_delivery procedure decomposes a datagram arriving from a remote IP into interface parameters and data and delivers them to the destination ULP. The data effects of this procedure are:

a. Data examined:

```

from_SNP.dtgm.source_addr
from_SNP.dtgm.destination_addr

```


MIL-STD-1777
12 August 1983

```

from_SNP.dtgm.protocol
from_SNP.dtgm.type_of_service
from_SNP.dtgm.total_length
from_SNP.dtgm.header_length
from_SNP.dtgm.data
from_SNP.dtgm.options

```

b. Data modified:

```

to_ULP.destination_addr      to_ULP.length
to_ULP.source_addr           to_ULP.data
to_ULP.protocol              to_ULP.options
to_ULP.type_of_service

```

```

to_ULP.destination_addr: = from_SNP.dtgm.destination_addr;
to_ULP.source_addr      : = from_SNP.dtgm.source_addr;
to_ULP.protocol         : = from_SNP.dtgm.protocol;
to_ULP.type_of_service : = from_SNP.dtgm.type_of_service;
to_ULP.length           : = from_SNP.dtgm.total_length -
                           from_SNP.dtgm.header_length*4;
to_ULP.data              : = from_SNP.dtgm.data;
to_ULP.options           : = from_SNP.dtgm.options;

```

NOTE: The format of the to_ULP elements is unspecified, allowing an implementor to assign data types for the interface parameters. If those data types differ from the IP header types, the assignment statements above become type conversions.

9.4.6.3.13 Route. The route procedure examines the destination address and options fields of an outbound datagram in to_SNP to determine a local destination address. The data effects of this procedure are:

a. Data examined:

```

to_SNP.dtgm.destination_addr
to_SNP.dtgm.options

```

b. Data modified:

```

to_SNP.local_destination_addr
to_SNP.dtgm.options

```

The procedure:

```

if (to_SNP.dtgm.options includes timestamp)
then
  if (the next timestamp field in to_SNP.dtgm.options.timestamp
      is available)
  then

```

MIL-STD-1777
12 August 1983

```
        --The timestamp or address/timestamp pair is inserted in
        --the next field in to_SNP.dtgm.options.timestamp.
    end if;
end if;

if (the network id field of destination matches the network id
    of the local subnet protocol )
then
    --Translate the REST field of destination into the subnetwork
    --address of the destination on this subnet.
    --implementation dependent action
else
    if (to_SNP.dtgm.options includes security)
    then
        --Find the appropriate gateway with security level equal to
        --the security level of to_SNP.dtgm.options.security. If
        --none exists, send error message.
    end if;

    if (to_SNP.dtgm.options includes loose source and record routing)
    then
        if (the network id field of next gateway in to_SNP.dtgm.option.
            loose_source matches the network of the local subnet
            protocol)
        then
            --The gateway address (as known in the environment into
            --which the datagram is being forwarded) replaces the
            --the network id field of next gateway in to_SNP.dtgm
            --.options.loose_source
        end if;
    end if;

    if (to_SNP.dtgm.options includes strict source and record routing)
    then
        if (the network id field of next gateway in to_SNP.dtgm.option.
            strict_source matches the network of the local subnet
            protocol)
        then
            --The gateway address (as known in the environment into
            --which the datagram is being forwarded) replaces the
            --the network id field of next gateway in to_SNP.dtgm
            --.options.strict_source.
        else
            --The datagram cannot be forwarded and error message sent.
        end if;
    end if;
end if;
```

MIL-STD-1777
12 August 1983

```
if (to_SNP.dtgm.options includes record routing)
then
  if (the next record route field in to_SNP.dtgm.options.record_
      routing is available)
  then
    --The gateway address (as known in the environment into
    --which the datagram is being forwarded) replaces the
    --next record route field in to_SNP.dtgm.options.record_
    --routing.
    end if;
  end if;
end if;

--Set the local destination interface parameter.

to_SNP.local_destination_addr := (subnetwork address found above)
```

MIL-STD-1777
12 August 1983

10. EXECUTION ENVIRONMENT REQUIREMENTS

10.1 Description. This section describes the facilities required of an execution environment for proper implementation and operation of the Internet Protocol. Throughout this document, the environmental model portrays each protocol as an independent process. Within this model, the execution environment must provide two facilities: interprocess communication and timing.

10.2 Interprocess communication. The execution environment must provide an interprocess communication facility to enable independent processes to exchange variable-length units of information, called messages. For IP's purposes, the IPC facility is not required to preserve the order of messages. IP uses the IPC facility to exchange interface parameters and data with upper layer protocols across its upper interface and the subnetwork protocol across the lower interface. Sections 6 and 8 specify these interfaces.

10.3 Timing. The execution environment must provide a timing facility that maintains a real-time clock with units no coarser than 1 millisecond. A process must be able to set a timer for a specific time period and be informed by the execution environment when the time period has elapsed. A process must also be able to cancel a previously set timer. Two IP mechanisms use the timing facility. The internet timestamp carries timing data in millisecond units. The reassembly mechanism uses timers to limit the lifetime of a datagram being reassembled. In the mechanism specification this facility is called TIMEOUT.

Custodians:

Army - CR
Navy - OM
Air Force - 90

Preparing Activity:

DCA - DC

(Project IPSC-0167-01)

Review Activities:

Army - SC, CR, AD
Navy - AS, YD, MC, OM, ND, NC, EC, SA
Air Force - 1, 11, 13, 17, 99, 90

Other Interest:

NSA - NS
TRI-TAC-TT

MIL-STD-1777
12 August 1983

APPENDIX A - DATA TRANSMISSION ORDER

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.

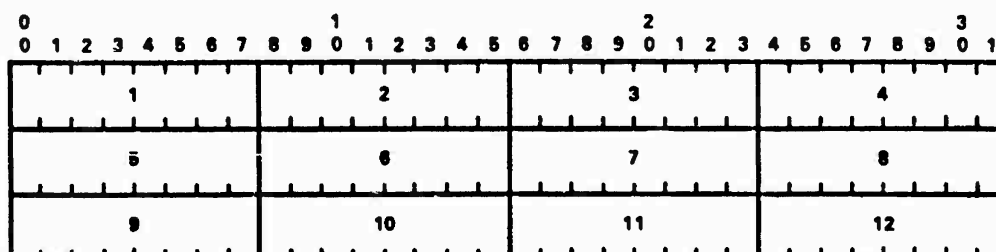


FIGURE 12. Transmission order of octets.

Whenever an octet represents a numeric quantity, the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).

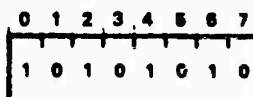


FIGURE 13. Significance of bits.

Similarly, whenever a multi-octet field represents a numeric quantity, the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

MIL-STD-1778
12 AUGUST 1983

MILITARY STANDARD

TRANSMISSION CONTROL PROTOCOL



NO DELIVERABLE DATA REQUIRED BY THIS DOCUMENT

MIL-STD-1778
NOTICE 1
26 October 1983

MILITARY STANDARD
TRANSMISSION CONTROL PROTOCOL

TO ALL HOLDERS OF MIL-STD-1778:

1. Table XII, page 111, "Net Deliver Event in an Estab State", and Table XIV, page 112, "Net Deliver Event in a CLOSE WAIT State", have been interchanged.
2. RETAIN THIS NOTICE PAGE AND INSERT BEFORE TABLE OF CONTENTS.

Custodians:

Army - CR
Navy - OM
Air Force - 90

Preparing Activity:

DCA-DC
(Project IPSC-0178-02)

Review Activities:

Army - SC, CR, AD
Navy - AS, YD, MC, OM, ND, NC, EC, SA
Air Force - 1, 11, 13, 17, 90, 99

Other Interest:

NSA-NS
TRI-TAC-TT

IPSC/SLHC/TCTS

U.S. GOVERNMENT PRINTING OFFICE: 1983 - 706-040/5648

MIL-STD-1778
12 August 1983

DEPARTMENT OF DEFENSE
WASHINGTON, D.C. 20301

Transmission Control Protocol

MIL-STD-1778

1. This Military Standard is approved for use by all Departments and Agencies of the Department of Defense.
2. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to: Defense Communications Agency, ATTN: J110, 1860 Wiehle Avenue, Reston, Virginia 22090, by using the self-addressed Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document, or by letter.

MIL-STD-1778
12 August 1983

FOREWORD

This document specifies the Transmission Control Protocol (TCP), a reliable connection-oriented transport protocol for use in packet-switched communication networks and internetworks. The document includes an overview with a model of operation, a description of services offered to users, and a description of the architectural and environmental requirements. The protocol service interfaces and mechanisms are specified using an extended state machine model.

MIL-STD-1778
12 August 1983

CONTENTS

		<u>Page</u>
Paragraph 1.	SCOPE - - - - -	1
1.1	Purpose - - - - -	1
1.2	Organization - - - - -	1
1.3	Application - - - - -	1
2.	REFERENCED DOCUMENTS- - - - -	2
2.1	Issues of documents - - - - -	2
2.2	Other publications- - - - -	2
3.	DEFINITIONS - - - - -	3
3.1	Definition of terms - - - - -	3
4.	GENERAL REQUIREMENTS- - - - -	7
4.1	Goal- - - - -	7
4.2	TCP defined - - - - -	7
4.3	Network layer provisions- - - - -	8
4.4	TCP design- - - - -	8
4.5	TCP mechanisms- - - - -	8
4.5.1	PAR mechanism - - - - -	8
4.5.2	Flow control mechanism- - - - -	9
4.5.3	Multiplexing mechanism- - - - -	9
4.6	ULP synchronization - - - - -	9
4.7	ULP modes - - - - -	9
4.8	Scenario- - - - -	10
4.9	Scenario notation - - - - -	10
5.	SERVICES PROVIDED TO UPPER LAYER- - - - -	14
5.1	Goal- - - - -	14
5.2	Service description - - - - -	14
5.2.1	Multiplexing service- - - - -	14
5.2.2	Connection management service - - - - -	14
5.2.2.1	Connection establishment - - - - -	14
5.2.2.2	Connection maintenance- - - - -	15
5.2.2.3	Connection termination- - - - -	15
5.2.3	Data transport service- - - - -	15
5.2.4	Capabilities provided to ULPs by TRP- - - - -	16
5.2.5	Error reporting service - - - - -	16
6.	UPPER LAYER SERVICE/INTERFACE SPECIFICATIONS- - - - -	17
6.1	Goal- - - - -	17
6.2	Interaction primitives- - - - -	17
6.2.1	Interaction primitive categories- - - - -	17
6.3	Service request primitives- - - - -	17
6.4	Parameter descriptions- - - - -	17
6.4.1	Unspecified passive open- - - - -	17
6.4.2	Fully specified passive open- - - - -	18
6.4.3	Active open - - - - -	18
6.4.4	Active open with data - - - - -	18

MIL-STD-1778
12 August 1983

CONTENTS - Continued

	<u>Page</u>
Paragraph 6.4.5	18
6.4.6	19
6.4.7	19
6.4.8	19
6.4.9	19
6.4.9.1	19
6.4.10	19
6.4.10.1	19
6.4.10.2	20
6.4.10.3	20
6.4.10.4	20
6.4.10.5	20
6.4.10.6	20
6.4.10.7	20
6.4.10.8	21
6.5	21
Extended state machine specification	
services provided to upper layer - - - - -	21
6.5.1	22
6.5.1.1	22
6.5.2	22
6.5.2.1	22
6.5.3	22
6.5.3.1	26
6.5.3.2	26
6.5.4	27
6.5.4.1	27
6.5.4.2	28
6.5.4.3	28
6.5.5	29
6.5.6	30
6.5.6.1	30
6.5.6.1.1	31
6.5.6.1.2	32
6.5.6.1.3	34
6.5.6.1.4	36
6.5.6.1.5	37
6.5.6.1.6	38
6.5.6.1.7	39
6.5.6.1.8	41
6.5.6.1.9	44
6.5.6.1.10	46

MIL-STD-1778
20 May 1983

CONTENTS - Continued

		<u>Page</u>
Paragraph 6.5.6.1.11	State A = closed, state B = closing- -	47
6.5.6.1.12	State A = active, state B = estab- lished- - - - -	48
6.5.6.1.13	State A = active, state B = closing- -	49
6.5.6.1.14	State A = passive, state B = estab- lished- - - - -	50
6.5.6.1.15	State A = passive, state B = closing- -	52
6.5.6.2	Decision functions - - - - -	53
6.5.6.2.1	Room_in (state vector name)- - - - -	53
6.5.6.2.2	Timeout_exceeded (sv*) - - - - -	53
6.5.6.3	Action procedures- - - - -	53
6.5.6.3.1	Add_to_send_queue (sv*)- - - - -	54
6.5.6.3.2	Assign_new_icn - - - - -	54
6.5.6.3.3	Error (local connection name, error description)- - - - -	54
6.5.6.3.4	Load security- - - - -	55
6.5.6.3.5	Initialize (sv*) - - - - -	55
6.5.6.3.6	Open_fail (local connection name)- - -	55
6.5.6.3.7	Open_id (local connection name, source port, source address, destination port, destination addr)- - - - -	55
6.5.6.3.8	Open_success (local connection name) -	56
6.5.6.3.9	Report_timeout (sv*)- - - - -	56
6.5.6.3.10	Requeue_oldest - - - - -	56
6.5.6.3.11	Terminate (local connection name, description)- - - - -	57
6.5.6.3.12	Sec_range_match- - - - -	57
6.5.6.3.13	Record_open_parameters (ULP identi- fier, open mode)- - - - -	57
6.5.6.3.14	Try_to_deliver - - - - -	58
7.	SERVICES REQUIRED FROM LOWER LAYER - - - - -	61
7.1	Goal - - - - -	61
7.2	Service descriptions - - - - -	61
7.2.1	Data transfer service- - - - -	61
7.2.2	Generalized network service- - - - -	61
7.2.3	Error reporting service- - - - -	61
8.	LOWER LAYER SERVICE/INTERFACE SPECIFICATIONS -	62
8.1	Goal - - - - -	62
8.2	Interaction primitives - - - - -	62
8.2.1	Service request primitives - - - - -	62
8.2.1.1	NET_SEND - - - - -	62
8.2.2	Service response primitives- - - - -	63
8.2.2.1	NET_DELIVER- - - - -	63
8.2.2.1.1	NET_DELIVER error reports- - - - -	64
8.3	Extended state machine specification of services required from lower layer- - - -	64

MIL-STD-1778
12 August 1983

CONTENTS - Continued

		<u>Page</u>
Paragraph 8.3.1	Machine instantiation identifier - - - - -	64
8.3.2	State diagram- - - - -	64
8.3.3	State vector - - - - -	64
8.3.4	Data structures- - - - -	64
8.3.4.1	To NET - - - - -	65
8.3.4.2	From NET - - - - -	65
8.3.5	Event list- - - - -	66
8.3.6	Events and actions- - - - -	66
8.3.6.1	EVENT = NET SEND (to NET) at time t - - -	66
8.3.6.2	EVENT = NULL- - - - -	67
9.	TCP ENTITY SPECIFICATION- - - - -	69
9.1	Goal- - - - -	69
9.2	Overview of TCP mechanisms- - - - -	69
9.2.1	Service support - - - - -	69
9.2.2	Background and terminology- - - - -	69
9.2.2.1	Sequence numbers- - - - -	69
9.2.2.1.1	Numbering scheme- - - - -	70
9.2.2.2	Connection sequence variables - - - - -	70
9.2.2.2.1	Send variables- - - - -	70
9.2.2.2.2	Send sequence space - - - - -	71
9.2.2.2.3	Receive variables - - - - -	71
9.2.2.2.4	Receive sequence space- - - - -	72
9.2.2.3	Current segment variables - - - - -	72
9.2.2.4	Connection states - - - - -	72
9.2.3	Flow control window - - - - -	73
9.2.3.1	Shrinking windows - - - - -	74
9.2.3.2	Zero windows- - - - -	74
9.2.3.3	Window updates with one-way data flow - -	75
9.2.3.4	Window management suggestions - - - - -	75
9.2.3.4.1	Window size vs. actual capacity - - - -	75
9.2.3.4.2	Small windows - - - - -	75
9.2.4	Duplicate and out-of-order data detection -	75
9.2.4.1	Incoming and unacceptable segments- - -	76
9.2.4.1.1	"In order" data acceptance- - - - -	76
9.2.4.1.2	"In window" data acceptance - - - - -	76
9.2.5	Positive acknowledgment with retransmission- - - - -	76
9.2.5.1	Acknowledgment generation - - - - -	77
9.2.5.2	ACK validation- - - - -	77
9.2.5.3	Retransmission queue removals - - - - -	77
9.2.5.4	Retransmission strategies - - - - -	77
9.2.5.5	Retransmission timeouts - - - - -	78
9.2.6	Checksum- - - - -	78
9.2.7	Push- - - - -	78
9.2.8	Urgent- - - - -	79
9.2.9	ULP timeout and ULP timeout action- - -	79
9.2.10	Security- - - - -	79

MIL-STD-1778
12 August 1983

CONTENTS - Continued

	<u>Page</u>
Paragraph 9.2.11	Precedence level- - - - - 80
9.2.12	Multiplexing- - - - - 80
9.2.13	Connection opening mechanisms - - - - - 81
9.2.13.1	Connection open requests- - - - - 81
9.2.13.1.1	Passive open request- - - - - 81
9.2.13.1.2	Active open request - - - - - 81
9.2.13.2	Three-way handshake - - - - - 81
9.2.13.2.1	Simplest handshake- - - - - 82
9.2.13.2.2	Examples of connection initiations- - - 82
9.2.13.2.2.1	Simultaneous connection initiation- - 82
9.2.13.2.2.2	Old duplicate SYN detection - - - - 83
9.2.13.2.2.3	Half-open connections - - - - - 83
9.2.13.2.2.4	Alternate case 1- - - - - 84
9.2.13.2.2.5	Alternate case 2- - - - - 85
9.2.13.3	Initial sequence number selection - - - 85
9.2.13.4	ISN generator - - - - - 85
9.2.14	Connection closing synchronization- - - 86
9.2.14.1	Close requests- - - - - 86
9.2.14.2	FIN exchange examples - - - - - 86
9.2.14.2.1	Case 1: local ULP initiates connection close- - - - - 86
9.2.14.2.2	Case 2: TCP receives FIN from remote TCP- - - - - 87
9.2.14.2.3	Case 3: ULPs close simultaneously- - - 87
9.2.14.3	Quiet time concept- - - - - 87
9.2.14.3.1	"Keep quiet" concept- - - - - 88
9.2.15	Resets- - - - - 88
9.2.15.1	Reset generation- - - - - 88
9.2.15.1.1	When connection does not exist- - - - 88
9.2.15.1.2	When connection is in any non- synchronized state - - - - - 89
9.2.15.1.3	When connection is in a synchronized state- - - - - 89
9.2.15.2	Reset processing- - - - - 89
9.3	TCP header format - - - - - 89
9.3.1	Source port - - - - - 90
9.3.2	Destination port- - - - - 90
9.3.3	Sequence number - - - - - 90
9.3.4	Acknowledgment number - - - - - 91
9.3.5	Data offset - - - - - 91
9.3.6	Reserved- - - - - 91
9.3.7	Control flags - - - - - 91
9.3.8	Window- - - - - 91
9.3.9	Checksum- - - - - 91
9.3.10	Urgent pointer- - - - - 92
9.3.11	Options - - - - - 92
9.3.11.1	Specific option definitions - - - - - 92
9.3.11.1.1	End of option list- - - - - 92

MIL-STD-1778
12 August 1983

CONTENTS - Continued

	<u>Page</u>
Paragraph 9.3.11.1.2	No-operation- - - - - 93
9.3.11.1.3	Maximum segment size- - - - - 93
9.3.12	Padding - - - - - 93
9.4	Extended state machine specification of
	TCP entity- - - - - 93
9.4.1	Machine instantiation identifier - - - - - 94
9.4.1.1	Socket pair identifier - - - - - 94
9.4.1.2	Local connection name- - - - - 94
9.4.2	State diagram- - - - - 94
9.4.3	State vector - - - - - 94
9.4.4	Data structures- - - - - 97
9.4.4.1	State vector - - - - - 97
9.4.4.2	From ULP - - - - - 98
9.4.4.3	To ULP - - - - - 99
9.4.4.4	To NET - - - - - 100
9.4.4.5	From NET - - - - - 100
9.4.4.6	Segment_type - - - - - 101
9.4.4.7	Supplemental type declarations - - - - - 101
9.4.5	Event list - - - - - 102
9.4.6	Events and actions - - - - - 103
9.4.6.1	Decision tables- - - - - 103
9.4.6.1.1	State = closed - - - - - 103
9.4.6.1.2	State = listen - - - - - 105
9.4.6.1.3	State = SYN_SENT - - - - - 106
9.4.6.1.4	State = SYN_RECV- - - - - 108
9.4.6.1.5	State = ESTAB- - - - - 109
9.4.6.1.6	State = CLOSE_WAIT - - - - - 111
9.4.6.1.7	State = closing- - - - - 113
9.4.6.1.8	State = FIN_WAIT1- - - - - 114
9.4.6.1.9	State = FIN_WAIT2- - - - - 116
9.4.6.1.10	State = last ACK - - - - - 117
9.4.6.1.11	State = TIME_WAIT- - - - - 118
9.4.6.2	Decision functions - - - - - 120
9.4.6.2.1	ACK_on - - - - - 120
9.4.6.2.2	ACK_status_test1 - - - - - 120
9.4.6.2.3	ACK_status_test2 - - - - - 121
9.4.6.2.4	Checksum_check - - - - - 121
9.4.6.2.5	FIN_ACK'd- - - - - 122
9.4.6.2.6	FIN_on - - - - - 123
9.4.6.2.7	FIN_seen - - - - - 123
9.4.6.2.8	Open_mode- - - - - 124
9.4.6.2.9	Sv_prec_vs_seq_prec- - - - - 124
9.4.6.2.10	Resources_suffice_open - - - - - 124
9.4.6.2.11	Resources_suffice_send - - - - - 125
9.4.6.2.12	RST_on - - - - - 125
9.4.6.2.13	Sec_match- - - - - 125
9.4.6.2.14	Sec_prec_allowed - - - - - 126
9.4.6.2.15	Sec_range_match- - - - - 126

MIL-STD-1778
12 August 1983

CONTENTS - Continued

	<u>Page</u>
Paragraph 9.4.6.2.16	Sec_prec_match - - - - - 127
9.4.6.2.17	Seq#_status- - - - - 127
9.4.6.2.18	SYN_on - - - - - 129
9.4.6.2.19	SYN_in_wndow - - - - - 129
9.4.6.2.20	Zero_recv_wndow- - - - - 129
9.4.6.3	Action_procedures- - - - - 130
9.4.6.3.1	Data management routines - - - - - 130
9.4.6.3.2	Accept - - - - - 130
9.4.6.3.3	Accept_policy- - - - - 131
9.4.6.3.4	Accept_strategy- - - - - 132
9.4.6.3.5	"In-order"_strategy- - - - - 132
9.4.6.3.6	Ack_policy - - - - - 132
9.4.6.3.7	Check_urg- - - - - 133
9.4.6.3.8	Compute_checksum - - - - - 133
9.4.6.3.9	Conn_open- - - - - 134
9.4.6.3.10	Deliver- - - - - 135
9.4.6.3.11	Deliver_policy - - - - - 137
9.4.6.3.12	Dispatch - - - - - 137
9.4.6.3.13	Dm_add_to_send - - - - - 138
9.4.6.3.14	Dm_add_to_recv - - - - - 138
9.4.6.3.15	Dm_copy_from_send- - - - - 138
9.4.6.3.16	Dm_remove_from_recv- - - - - 139
9.4.6.3.17	Dm_remove_from_send- - - - - 139
9.4.6.3.18	Error- - - - - 139
9.4.6.3.19	Format_net_params- - - - - 140
9.4.6.3.20	Gen_id - - - - - 140
9.4.6.3.21	Gen_isn- - - - - 141
9.4.6.3.22	Gen_lcn- - - - - 141
9.4.6.3.23	Gen_syn- - - - - 141
9.4.6.3.24	Load_security- - - - - 143
9.4.6.3.25	New_allocation - - - - - 143
9.4.6.3.26	Open - - - - - 144
9.4.6.3.27	Openfail - - - - - 146
9.4.6.3.28	Part_reset - - - - - 146
9.4.6.3.29	Raise_prec - - - - - 147
9.4.6.3.30	Record_syn - - - - - 147
9.4.6.3.31	Report_timeout (sv*)- - - - - 148
9.4.6.3.32	Requeue_oldest (sv*)- - - - - 149
9.4.6.3.33	Reset- - - - - 149
9.4.6.3.34	Reset_self - - - - - 150
9.4.6.3.35	Restart_time_wait- - - - - 151
9.4.6.3.36	Retransmit - - - - - 151
9.4.6.3.37	Retransmit_policy- - - - - 153
9.4.6.3.37.1	Retransmission strategy- - - - - 153
9.4.6.3.38	Save_fin - - - - - 154
9.4.6.3.39	Save_send_data - - - - - 154
9.4.6.3.40	Send_ack - - - - - 155

MIL-STD-1778
12 August 1983

CONTENTS - Continued

		<u>Page</u>
Paragraph 9.4.6.3.41	Send_fin - - - - -	155
9.4.6.3.42	Send_new_data- - - - -	156
9.4.6.3.43	Send_policy- - - - -	157
9.4.6.3.44	Set_fin - - - - -	158
9.4.6.3.45	Start_time_wait- - - - -	158
9.4.6.3.46	Update - - - - -	159
10.	EXECUTION ENVIRONMENT REQUIREMENTS- - - - -	160
10.1	Introduction- - - - -	160
10.2	Inter-process communication - - - - -	160
10.3	Timing- - - - -	160

APPENDICES

Appendix A	RETRANSMISSION STRATEGY EFFECTIVENESS - - - - -	161
B	DYNAMIC RETRANSMISSION TIMER COMPUTATION- - - - -	162
C	ALTERNATIVES IN SERVICE INTERFACE PRIMITIVES- - - - -	163

MIL-STD-1778
12 August 1983

CONTENTS - Continued

		<u>Page</u>
FIGURES		
Figure 1	Example host protocol hierarchy - - - - -	7
2A	A simple connection opening - - - - -	10
2B	A simple connection opening - - - - -	11
3A	Two-way data transfer - - - - -	11
3B	Two-way data transfer - - - - -	12
4A	A graceful connection close - - - - -	12
4B	A graceful connection close - - - - -	13
5	Split state model of TCP services - - - - -	21
6	Composite TCP service state machine diagram - -	23
7	TCP local service state machine summary - - -	24
8	Complex sec_range structure - - - - -	26
9	TCP header format - - - - -	90
10	End of option list code - - - - -	92
11	No-operation option code - - - - -	93
12	Maximum segment size option - - - - -	93
13	TCP entity state summary - - - - -	95
14	Checksum check function - - - - -	122
15	Compute checksum procedure - - - - -	134
16	TCP local service state machine summary - - -	164

TABLES		
Table I	Active_open event in a closed state - - - - -	103
II	Active_open with data event in a closed state -	104
III	Full_passive_open event in a closed state - - -	104
IV	Unspecified_passive_open event in a closed state - - - - -	104
V	Net_deliver event in a closed state - - - - -	105
VI	Net_deliver event in a listen state - - - - -	106
VII	Close or abort event in a SYN_SENT state - - -	106
VIII	Net_deliver event in a SYN_SENT state - - - -	107
IX	Send event in a SYN_RECVD state - - - - -	108
X	Net_deliver event in a SYN_RECVD state - - - -	109
XI	Send event in an estab state - - - - -	110
XII	Net_deliver event in an estab state - - - - -	111
XIII	Send event in a CLOSE_WAIT state - - - - -	111
XIV	Net_deliver event in a CLOSE_WAIT state - - - -	112
XV	Net_deliver event in a closing state - - - - -	114
XVI	Net_deliver event in a FIN_WAIT1 state - - - -	115
XVII	Net_deliver event in a FIN_WAIT2 state - - - -	117
XVIII	Net_deliver event in a LAST_ACK state - - - -	118
XIX	Net_deliver event in a TIME_WAIT state - - - -	119

MIL-STD-1778
12 August 1983

1. SCOPE

1.1 Purpose. This standard establishes criteria for the Transmission Control Protocol (TCP), a reliable connection-oriented transport protocol for use in packet-switched and other communication networks and interconnected sets of such networks.

1.2 Organization. This standard is organized into ten paragraphs. Beginning with paragraph 4, the TCP's role is established in the evolving DoD protocol architecture and the TCP's major services and mechanisms are also introduced. Paragraphs 5 and 6 more formally specify the services TCP offers to upper layer protocols and the interface through which those services are accessed. Similarly, paragraphs 7 and 8 specify the services required of the lower layer protocol and the lower interface. Paragraph 9 specifies the mechanisms supporting the TCP services and paragraph 10 outlines the functionality required of the execution environment for successful TCP operation.

1.3 Application. The Transmission Control Protocol (TCP) and the Internet Protocol (IP) are mandatory for use in all DoD packet switching networks which connect or have the potential for utilizing connectivity across network or subnetwork boundaries. Network elements (hosts, front-ends, bus interface units, gateways, etc.) within such networks which are to be used for inter-netting shall implement TCP/IP. The term network as used herein includes Local Area Networks (LANs) but not integrated weapons systems. Use of TCP/IP within LANs is strongly encouraged particularly where a need is perceived for equipment interchangeability or network survivability. Use of TCP/IP in weapons systems is also encouraged where such usage does not diminish network performance.

MIL-STD-1778
12 August 1983

2. REFERENCED DOCUMENTS

2.1 Issues of documents. The following documents of the issue in effect on date of invitation for bids or request for proposal, form a part of this standard to the extent specified herein. (The provisions of this paragraph are under consideration.)

2.2 Other publications. The following documents form a part of this standard to the extent specified herein. Unless otherwise indicated, the issue in effect on date of invitation for bids or request for proposals shall apply. (The provisions of this paragraph are under consideration.)

MIL-STD-1778
12 August 1983

3. DEFINITIONS

3.1 Definition of terms. The definition of terms used in this standard shall comply with FED-STD-1037. Terms and definitions unique to MIL-STD-1778 are contained herein.

- a. Acknowledgment Number. A 32-bit field of the TCP header containing the next sequence number expected by the sender of the segment.
- b. ACK. Acknowledgment flag: a control bit in the TCP header indicating that the acknowledgment number field is significant for this segment.
- c. Checksum. A 16-bit field of the TCP header carrying the one's complement based checksum of both the header and data in the segment.
- d. connection. A logical communication path identified by a pair of sockets.
- e. datagram. A self-contained package of data carrying enough information to be routed from source to destination without reliance on earlier exchanges between source or destination and the transporting network.
- f. datagram service. A datagram, defined above, delivered in such a way that the receiver can determine the boundaries of the datagram as it was entered by the source. A datagram is delivered with high probability to the desired destination, but it may possibly be lost. The sequence in which datagrams are entered into the network by a source is not necessarily preserved upon delivery at the destination.
- g. Data Offset. A TCP header field containing the number of 32-bit words in the TCP header.
- h. Destination Address. The destination address, usually the network and host identifiers. Although not carried in the TCP header, this value is passed to and received from the network protocol entity with each segment.
- i. Destination Port. The TCP header field containing a 2-octet value identifying the destination upper level protocol of a segment's data.
- j. EFIP. Electronic File Transfer Protocol. Electronic mail.
- k. FIN. A control bit of the TCP header indicating that no more data will be sent by the sender.
- l. FTP. File Transfer Protocol

MIL-STD-1778
12 August 1983

- m. header. The collection of control information transmitted with data between peer entities.
- n. host. A computer, particularly a source or destination of messages from the point of view of the communication network.
- o. Identification. A value passed with each segment to the network protocol entity (Internet Protocol). This identifying value assigned by the sending TCP aids in assembling the fragments of a datagram.
- p. internetwork. A set of interconnected subnetworks.
- q. internet address. A four octet (32 bit) source or destination address composed of a Network field and a Local Address field.
- r. internet datagram. The package exchanged between a pair of IP modules. It is made up of an internet header and a data portion.
- s. IP. Internet Protocol
- t. ISN. The Initial Sequence Number. The first sequence number used for either sending or receiving on a connection. It is selected on a clock based procedure.
- u. local network. The network directly attached to host or gateway.
- v. MSL. Maximum Segment Lifetime, the time a TCP segment can exist in the internet-work system. Arbitrarily defined to be 2 minutes.
- w. Options. The optional set of fields at the end of the TCP header used in a SYN segment to carry the maximum segment size acceptable to the sender.
- x. packet network. A network based on packet-switching technology. Messages are split into small units (packets) to be routed independently on a store and forward basis. This packetizing pipelines packet transmission to effectively use circuit bandwidth.
- y. Padding. A header field inserted after option fields to ensure that the data portion begins on a 32-bit word boundary. The padding field value is zero.
- z. PUSH. A control bit of the TCP header occupying no sequence space, indicating that this segment contains data that must be pushed through to the receiving ULP.
- aa. push service. A service provided by TCP to the upper level protocols. A push directs TCP to segment, send, and deliver data received up to that point as soon as flow control permits.

MIL-STD-1778
12 August 1983

- bb. receive next sequence number. The next sequence number a TCP is expecting to receive.
- cc. receive window. This represents the sequence numbers a TCP is willing to receive. Thus, the TCP considers that segments overlapping the range $RECV_NEXT$ to $RECV_NEXT + RECV_WND - 1$ carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.
- dd. Reserved. A 6-bit field of the TCP header that is not currently used but must be zero.
- ee. RST. A control bit of the TCP header indicating that the connection associated with this segment is to be terminated.
- ff. segment. The unit of data exchanged by TCP modules. This term may also be used to describe the unit of exchange between any transport protocol modules.
- gg. segment length. The amount of sequence number space occupied by segment, including any controls which occupy sequence space.
- hh. send sequence. This is the next sequence number the TCP will use to send data on the connection. It is initially selected from an initial sequence number curve (ISN) and is incremented for each octet of data or sequenced control transmitted.
- ii. send window. This represents the sequence numbers which the remote TCP is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP. The range of new sequence numbers which may be emitted by a TCP lies between $SEND_NEXT$ and $SEND_UNA + SEND_WND - 1$. (Retransmissions of sequence numbers between $SEND_UNA$ and $SEND_NEXT$ are expected, of course.)
- jj. Sequence Number. A 32-bit field of the TCP header containing the sequence number of the 1) a sequenced control flag (if present), or 2) the first byte of data (if present), or, 3) for empty segments, the sequence number of the next data octet to be sent.
- kk. socket. An address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port.
- ll. Source Port. The TCP header field containing a 2-octet value identifying the source upper level protocol of a segment's data.
- mm. TCP segment. The package exchanged between TCP modules made up of the TCP header and a text portion (which may be empty).
- nn. UDP. User Datagram Protocol

MIL-STD-1778
12 August 1983

- oo. ULP. Upper Level Protocol: any protocol above TCP in the layered protocol hierarchy that uses TCP. This term includes presentation layer protocols, session layer protocols, and user applications.
- pp. Urgent Pointer. A TCP header field containing a positive offset to the sequence number of the segment indicating the position of urgent data in the connection's data stream. This field is valid only when the URG flag is on.
- qq. URG. A control bit of the TCP header indicating that the urgent field contains a valid pointer to urgent information in the connection's data stream.
- rr. Window. A 2-octet field of the TCP header indicating the number of data octets (relative to the acknowledgment number in the header) that the segment sender is currently willing to accept.

MIL-STD-1778
12 August 1983

4. GENERAL REQUIREMENTS

4.1 Goal. One goal of this standard is to avoid assuming a particular system configuration. As a practical matter, the distribution of protocol layers to specific hardware configurations will vary. For example, many computer systems are connected to networks via front-end computers which house TCP and lower layer protocol software. Although appearing to focus on TCP implementations which are co-resident with the upper and lower layer protocols, this specification can apply to any configuration given appropriate inter-layer protocols to bridge hardware boundaries.

4.2 TCP defined. TCP is designed to provide reliable communication between pairs of processes in logically distinct hosts on networks and sets of interconnected networks. Thus, TCP serves as the basis for DoD-wide inter-process communication in communication systems. TCP will operate successfully in an environment where the loss, damage, duplication, or disorder data, and network congestion can occur. This robustness in spite of unreliable communications media makes TCP well suited to support military, governmental, and commercial applications. TCP appears in the DoD protocol hierarchy at the transport layer. Here, TCP provides connection-oriented data transfer that is reliable, ordered, full-duplex, and flow controlled. TCP is designed to support a wide range of upper layer protocols (ULPs). The ULPs can channel continuous streams of data through TCP for delivery to peer ULPs. TCP breaks the streams into portions which are encapsulated together with appropriate addressing and control information to form a segment--the unit of exchange between peer TCPs. In turn, TCP passes segments to the network layer for transmission through the communication system to the peer TCP.

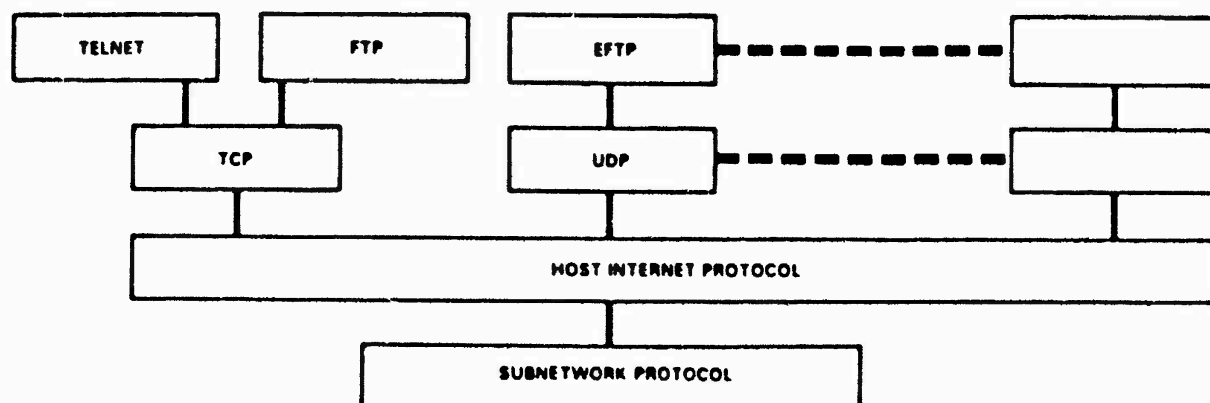


FIGURE 1. Example host protocol hierarchy.

MIL-STD-1778
12 August 1983

4.3 Network layer provisions. The network layer provides for data transfer between hosts attached to a communication system. Such systems may range from a single network to interconnected sets of networks forming an internetwork. The minimum required data transfer service is limited; data may be lost, duplicated, misordered, or damaged in transit. As part of the transfer service though, the network layer must provide global addressing, handle routing, and hide network-specific characteristics. As a result, upper layer protocols (including TCP) using the network layer may operate above a wide spectrum of subnetwork systems ranging from hard-wire connections to packet-switched or circuit-switched subnets. Additional services the network layer may provide include selectable levels of transmission quality such as precedence, reliability, delay, and throughput. The network layer also allows data labelling, needed in secure environments, to associate security information with data.

4.4 TCP design. TCP was specifically designed to operate above the Internet Protocol (IP) which supports the interconnection of networks. IP's internet datagram service provides the functionality described above. Originally, TCP and IP were developed as a single protocol providing resource sharing across different packet networks. The need for other transport protocols to use IP's services led to their specification as two distinct protocols.

4.5 TCP mechanisms. TCP builds its services on top of the network layer's potentially unreliable ones with mechanisms such as error detection, positive acknowledgments, sequence numbers, and flow control. These mechanisms require certain addressing and control information to be initialized and maintained during data transfer. This collection of information is called a TCP connection. The following paragraphs describe the purpose and operation of the major TCP mechanisms.

4.5.1 PAR mechanism. TCP uses a positive acknowledgement with retransmission (PAR) mechanism to recover from the loss of a segment by the lower layers. The strategy with PAR is for a sending TCP to retransmit a segment at timed intervals until a positive acknowledgement is returned. The choice of retransmission interval affects efficiency. An interval that is too long reduces data throughput while one that is too short floods the transmission media with superfluous segments. In TCP, the timeout is expected to be dynamically adjusted to approximate the segment round-trip time plus a factor for internal processing, otherwise performance degradation may occur. TCP uses a simple checksum to detect segments damaged in transit. Such segments are discarded without being acknowledged. Hence, damaged segments are treated identically to lost segments and are compensated for by the PAR mechanism. TCP assigns sequence numbers to identify each octet (an eight bit byte) of the data stream. These enable a receiving TCP to detect duplicate and out-of-order segments. Sequence numbers are also used to extend the PAR mechanism by allowing a single acknowledgment to cover many segments worth of data. Thus, a sending TCP can still send new data although previous data has not been acknowledged.

MIL-STD-1778
12 August 1983

4.5.2 Flow control mechanism. TCP's flow control mechanism enables a receiving TCP to govern the amount of data dispatched by a sending TCP. The mechanism is based on a "window" which defines a contiguous interval of acceptable sequence numbered data. As data is accepted, TCP slides the window upward in the sequence number space. This window is carried in every segment enabling peer TCPs to maintain up-to-date window information.

4.5.3 Multiplexing mechanism. TCP employs a multiplexing mechanism to allow multiple ULPs within a single host and multiple processes in a ULP to use TCP simultaneously. This mechanism associates identifiers, called ports, to ULP's processes accessing TCP services. A ULP connection is uniquely identified with a socket, the concatenation of a port and an internet address. Each connection is uniquely named with a socket pair. This naming scheme allows a single ULP to support connections to multiple remote ULPs. ULPs which provide popular resources are assigned permanent sockets, called well-known sockets.

4.6 ULP synchronization. When two ULPs wish to communicate, they instruct their TCPs to initialize and synchronize the mechanism information on each to open the connection. However, the potentially unreliable network layer can complicate the process of synchronization. Delayed or duplicate segments from previous connection attempts might be mistaken for new ones. A handshake procedure with clock based sequence numbers is used in connection opening to reduce the possibility of such false connections. In the simplest handshake, the TCP pair synchronizes sequence numbers by exchanging three segments, thus the name three-way handshake. The scenario following the overview depicts this exchange. The procedure will be discussed more fully in the mechanism descriptions, Paragraph 9.2.

4.7 ULP modes. A ULP can open a connection in one of two modes, passive or active. With a passive open a ULP instructs its TCP to be "receptive" to connections with other ULPs. With an active open a ULP instructs its TCP to actively initiate a three-way handshake to connect to another ULP. Usually, an active open is targeted to a passive open. This active/passive model supports server-oriented applications where a permanent resource, such as a data-base management process, can always be accessed by remote users. However, the three-way handshake also coordinates two simultaneous active opens to open a connection. Over an open connection, the ULP-pair can exchange a continuous stream of data in both directions. Normally, TCP transparently groups the data into TCP segments for transmission at its own convenience. However, a ULP can exercise a "push" service to force TCP to package and send data passed up to that point without waiting for additional data. This mechanism is intended to prevent possible deadlock situations where a ULP waits for data internally buffered by TCP. For example, an interactive editor might wait forever for a single input line from a terminal. A push will force data through the TCPs to the waiting process. TCP also provides a means for a sending ULP to indicate to a receiving ULP that "urgent" data appears in the upcoming data stream. This urgent mechanism can support, for example, interrupts or breaks. When data exchange is complete the connection can be closed by either ULP to free TCP resources for other connections. Connection closing can happen in two ways. The first,

MIL-STD-1778
12 August 1983

called a graceful close, is based on the three-way handshake procedure to complete data exchange and coordinate closure between the TCPs. The second, called an abort, does not allow coordination and may result in loss of unacknowledged data.

4.8 Scenario. The following scenario provides a walk-through of a connection opening, data exchange, and a connection closing as might occur between the data base management process and user mentioned above. The scenario glosses over many details to focus on the three-way handshake mechanism in connection opening and closing, and the positive acknowledgment with retransmission mechanism supporting reliable data transfer. Although not pictured, the network layer transfers the information between the TCPs. For the purposes of this scenario, the network layer is assumed not to damage, lose, duplicate, or change the order of data unless explicitly noted. The scenario is organized into three parts:

- a. A simple connection opening in steps 1-7.
- b. Two-way data transfer in steps 8-17.
- c. A graceful connection close in steps 18-25.

4.9 Scenario notation. The following notation is used in the diagrams:

<-- SEQ# 200 <--	depicts information exchange
--> ACK# 201 -->	between peer TCPs
: △	depicts information passing
SEND DATA :	across the interface between
: DELIVER DATA	a ULP and its TCP
v :	

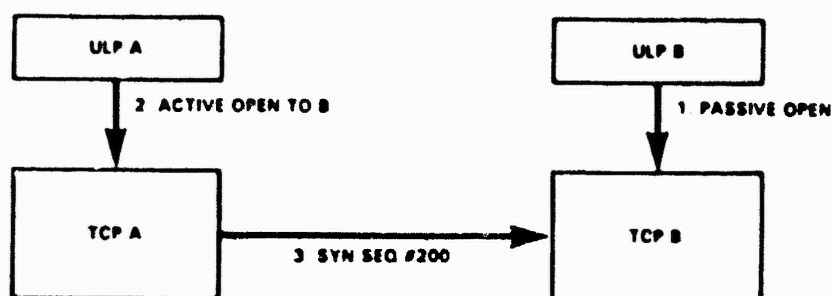


FIGURE 2A. A simple connection opening.

- a. ULP B (the DB manager) issues a PASSIVE OPEN to TCP B to prepare for connection attempts from other ULPs in the system.
- b. ULP A (the user) issues an ACTIVE OPEN to open a connection to ULP B.
- c. TCP A sends a segment to TCP B with an OPEN control flag, called a SYN, carrying the first sequence number (shown as SEQ#200) it will use for data sent to B.

MIL-STD-1778
12 August 1983

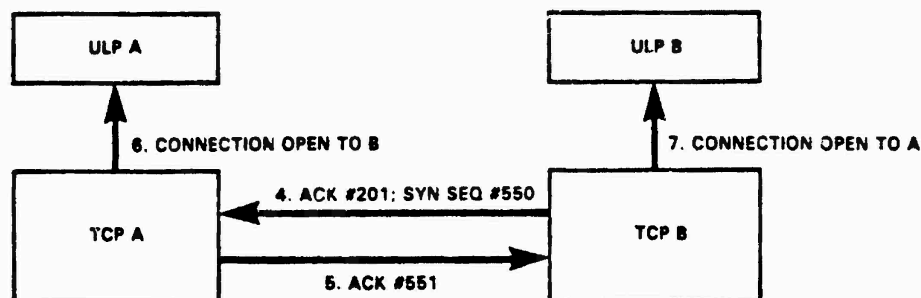


FIGURE 2B. A simple connection opening.

- d. TCP B responds to the SYN by sending a positive acknowledgment, or ACK, marked with next sequence number expected from TCP A. In the same segment, TCP B sends its own SYN with the first sequence number for its data (SEQ#550).
- e. TCP A responds to TCP B's SYN with an ACK showing the next sequence number expected from B.
- f. TCP A now informs ULP A that a connection is open to ULP B.
- g. Upon receiving the ACK, TCP B informs ULP B that a connection has been opened to ULP A.

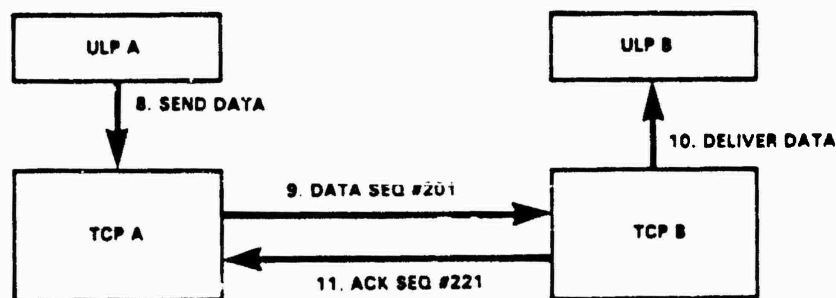


FIGURE 3A. Two-way data transfer.

- h. ULP A passes 20 octets of data to TCP A for transfer across the open connection to ULP B.
- i. TCP A packages the data in a segment marked with current "A" sequence number.
- j. After validating the sequence number, TCP B accepts the data and delivers it to ULP B.
- k. TCP B acknowledges all 20 octets of data with the ACK set to the sequence number of the next data octet expected.

MLL-STD-1718
12 August 1983

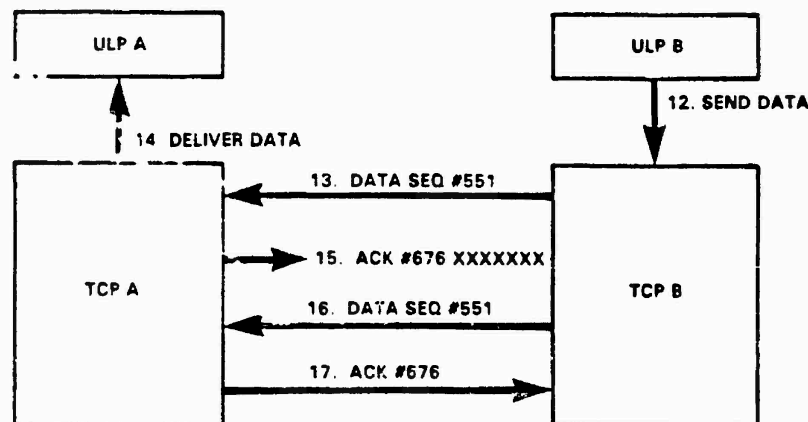


FIGURE 3B. Two-way data transfer.

- l. ULP B passes 125 bytes of data to TCP B for transfer to ULP A.
- m. TCP B packages the data in a segment marked with the "B" sequence number.
- n. TCP A accepts the segment and delivers the data to ULP A.
- o. TCP A returns an ACK of the received data marked with the sequence number of the next expected data octet. However, the segment is lost by the network and never arrives at TCP B.
- p. TCP B times out waiting for the lost ACK and retransmits the segment. TCP A receives the retransmitted segment, but discards it because the data from the original segment has already been accepted. However, TCP A re-sends the ACK.
- q. TCP B gets the second ACK.

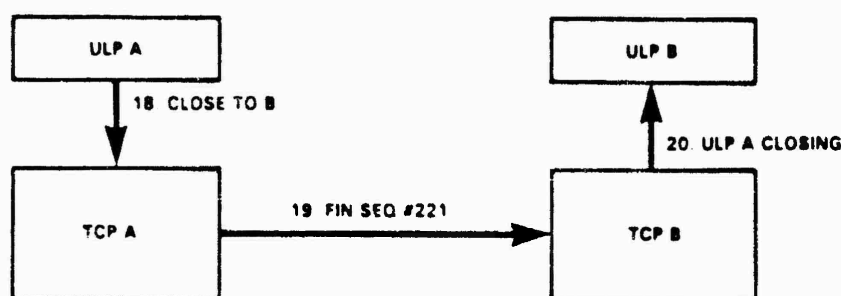


FIGURE 4A. A graceful connection close.

- r. ULP A closes its half of the connection by issuing a CLOSE to TCP A.

MIL-STD-1778
12 August 1983

- s. TCP A sends a segment marked with a CLOSE control flag, called a FIN, to inform TCP B that ULP A will send no more data.
- t. TCP B gets the FIN and informs ULP B that ULP A is closing.

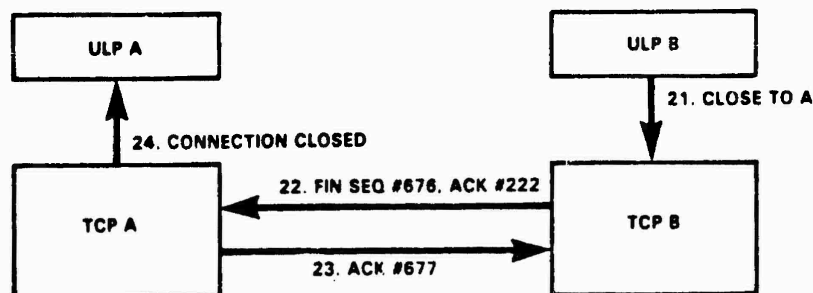


FIGURE 4B. A graceful connection close.

- u. ULP B completes its data transfer and closes its half of the connection. TCP B sends an ACK of the first FIN and its own FIN to TCP A to show ULP B's closing. TCP A gets the FIN and the ACK, then responds with an ACK to TCP B. TCP A informs ULP A that the connection is closed. (Not pictured) TCP B receives the ACK from TCP A and informs ULP B that the connection is closed.

MIL-STD-1778
12 August 1983

5. SERVICES PROVIDED TO UPPER LAYER

5.1 Goal. This section describes the services offered by the Transmission Control Protocol to upper layer protocols (ULPs). The goals of this section are to provide the motivation for protocol mechanisms and to provide ULPs with a definition of the functions provided by this protocol. The services provided by TCP can be organized as follows:

- a. multiplexing service
- b. connection management services
- c. data transport service
- d. error reporting service

5.2 Service description. A description of each service follows.

5.2.1 Multiplexing service. TCP shall provide services to multiple pairs of processes within upper layer protocols. A process within a ULP using TCP services shall be identified with a "port". A port, when concatenated with an internet address, forms a socket which uniquely names a ULP throughout the internet. TCP shall use the pair of sockets corresponding to a connection to differentiate between multiple users.

5.2.2 Connection management service. TCP shall provide data transfer capabilities, called connections, between pairs of upper layer protocols. A connection provides a communication channel between two ULPs. Characteristics of data transfer are specified in the data transfer service description. Connection management can be broken into three phases: connection establishment, connection maintenance, and connection termination.

5.2.2.1 Connection establishment. TCP shall provide a means to open connections between ULP-pairs. Connections are endowed with certain properties that apply for the lifetime of the connection. These properties, including security and precedence levels, are specified by the ULPs at connection opening. Connections can be opened in one of two modes: active or passive. TCP shall provide a means for a ULP to actively initiate a connection to another ULP uniquely named with a socket. TCP shall establish a connection to the named ULP if:

- a. no connection between the two named sockets already exists,
- b. internal TCP resources are sufficient,
- c. the other ULP exists, and has simultaneously executed a matching active open to this ULP, or previously executed a matching passive open, or previously executed a "global" matching passive open. TCP shall provide a means for a ULP to listen for and respond to active opens from correspondent ULPs. Correspondent ULPs are named in one of two ways:
- d. fully specified: A ULP is uniquely named by a socket. A connection is established when a matching active open is executed (as described above) by the named ULP.

MIL-STD-1778
12 August 1983

- e. unspecified: No socket is provided. A connection is established with any ULP executing a matching active open naming this ULP.

5.2.2.2 Connection maintenance. TCP shall maintain established connections supporting the data transfer service described in paragraph 5.2.3. And, TCP shall provide a means for a ULP to acquire current connection status with regard to connection name, data transfer progress, and connection qualities.

5.2.2.3 Connection termination. TCP shall provide a means to terminate established connections and nullify connection attempts. Established connections can be terminated in two ways:

- a. Graceful Close: Both ULPs close their side of the duplex connection, either simultaneously or sequentially, when data transfer is complete. TCP shall coordinate connection termination and prevent loss of data in transit as promised by the data transfer service.
- b. Abort: One ULP independently forces closure of the connection. TCP shall not coordinate connection termination. Any data in transit may be lost.

5.2.3 Data transport service. TCP shall provide data transport over established connections between ULP-pairs. The data transport is full-duplex, timely, ordered, labelled with security and precedence levels, flow controlled, and error-checked. A more detailed description of each of the data transport characteristics follows.

- a. full-duplex: TCP shall support simultaneous bi-directional data flow between the correspondent ULPs.
- b. timely: When system conditions prevent timely delivery, as specified by the user timeout, TCP shall notify the local ULP of service failure and subsequently terminate the connection.
- c. ordered: TCP shall deliver data to a destination ULP in the same sequence as it was provided by the source ULP.
- d. labelled: TCP shall associate with each connection the security and precedence levels supplied by the ULPs during connection establishment. When this information is not provided by the ULP-pair, TCP shall assume default levels. TCP shall establish a connection between a ULP-pair only if the security/compartments information exactly matches. If the precedence levels do not match during connection, the higher precedence level is associated with the connection.
- e. flow controlled: TCP shall regulate the flow of data across the connection to prevent, among other things, internal TCP congestion leading to service degradation and failure.

MIL-STD-1778
12 August 1983

- f. error checked: TCP shall deliver data that is free of errors within the probabilities supported by a simple checksum.

5.2.4 Capabilities provided to ULPs by TCP. TCP shall provide two capabilities to ULPs concerning data transfer over an established connection: data stream push and urgent data signalling.

- a. data stream push: TCP shall transmit any waiting data up to and including the indicated data portions to the receiving TCP without waiting for additional data. The receiving TCP shall deliver the data to the receiving ULP in the same manner.
- b. urgent data signalling: TCP shall provide a means for a sending ULP to inform a receiving ULP of the presence of significant, or "urgent," data in the upcoming data stream.

5.2.5 Error reporting service. TCP shall report service failure stemming from catastrophic conditions in the internetwork environment for which TCP cannot compensate.

MIL-STD-1778
12 August 1983

6. UPPER LAYER SERVICE/INTERFACE SPECIFICATIONS

6.1 Goal. The goal of this section is to specify the TCP services provided to upper layer protocols and the interface through which these services are accessed. The first part defines the interaction primitives and interface parameters for the upper interface. The second part contains the extended state machine specification of the upper layer services and interaction discipline.

6.2 Interaction primitives. An interaction primitive defines the information exchanged between two adjacent protocol layers. Primitives are grouped into two classes based on the direction of information flow. Information passed downward, in this case from a ULP to TCP, is called a service request primitive. Information passed upward, from TCP to the ULP, is called a service response primitive. Interaction primitives need not occur in pairs. That is, a service request does not necessarily elicit a service "response"; a service "response" may occur independently of a service request.

6.2.1 Interaction primitive categories. The information associated with an interaction primitive falls into two categories: parameters and data. Parameters describe the data and indicate how it is to be treated. The data itself is neither examined nor modified. The format of the parameters and data is implementation dependent and therefore not specified. TCP implementations may have different interaction primitives imposed by the execution environment or system design factors. In those cases, the primitives can be modified to include more information or additional primitives can be defined to satisfy system requirements. However, all TCPs must provide at least the information found in the interaction primitives specified below to guarantee that all TCP implementations can support the same protocol hierarchy. Additional primitives that affect the protocol mechanisms may not be used.

6.3 Service request primitives. The TCP service request primitives enable connection establishment, data transfer, and connection termination. The request primitives are:

- a. Unspecified Passive Open,
- b. Fully Specified Passive Open,
- c. Active Open,
- d. Active Open With Data,
- e. Send,
- f. Allocate,
- g. Close,
- k. Abort, and
- i. Status.

6.4 Parameter descriptions. A description and list of parameters for each service request follows. Optional service request parameters are followed by "[optional]."

6.4.1 Unspecified passive open. This service request primitive allows a ULP to listen for and respond to connection attempts from an unnamed ULP at a specified security and precedence level. TCP accepts in an Unspecified Passive Open at least the following information:

MIL-STD-1778
12 August 1983

- a. source port
- b. ULP timeout [optional]
- c. ULP timeout action [optional]
- d. precedence [optional]
- e. security_range [optional]

6.4.2 Fully specified passive open. This service request primitive allows a ULP to listen for and respond to connection attempts from a fully named ULP at a particular security and precedence level. TCP accepts in a Fully Specified Passive Open at least the following information:

- a. source port
- b. destination port
- c. destination address
- d. ULP timeout [optional]
- e. ULP timeout action [optional]
- f. precedence [optional]
- g. security_range [optional]

6.4.3 Active open. This service request primitive allows a ULP to initiate a connection attempt to a named ULP at a particular security and precedence level. TCP accepts in an Active Open at least the following information:

- a. source port
- b. destination port
- c. destination address
- d. ULP timeout [optional]
- e. ULP timeout action [optional]
- f. precedence [optional]
- g. security [optional]

6.4.4 Active open with data. This service request primitive allows a ULP to initiate a connection attempt to a named ULP at a particular security and precedence level accompanied by the specified data. TCP accepts in an Active Open With Data at least the following information:

- a. source port
- b. destination port
- c. destination address
- d. ULP timeout [optional]
- e. ULP timeout action [optional]
- f. precedence [optional]
- g. security [optional]
- h. data
- i. data length
- j. PUSH flag
- k. URGENT flag

6.4.5 Send. This service request primitive causes data to be transferred across the named connection. TCP accepts in a Send at least the following information:

MIL-STD-1778
12 August 1983

- a. local connection name
- b. data
- c. data length
- d. PUSH flag
- e. URGENT flag
- f. ULP timeout [optional]
- g. ULP timeout_action [optional]

6.4.6 Allocate. This service request primitive allows a ULP to issue TCP an incremental allocation for receive data. The parameter, data length, is defined in single octet units. This quantity is the additional number of octets which the receiving ULP is willing to accept. TCP accepts in an Allocate at least the following information:

- a. local connection name
- b. data length

6.4.7 Close. This service request primitive allows a ULP to indicate that it has completed data transfer across the named connection. TCP accepts in a Close at least the following information:

- a. local connection name

6.4.8 Abort. This service request primitive allows a ULP to indicate that the named connection is to be immediately terminated. TCP accepts in an Abort at least the following information:

- a. local connection name

6.4.9 Status. This service request primitive allows a ULP to query for the current status of the named connection. TCP accepts in a Status at least the following information.

- a. local connection name

6.4.9.1 Status responses. TCP returns the requested status information in a Status Response, defined in Section 6.4.10.7.

6.4.10 Service response primitives. Several service response primitives are provided to enable TCP to inform the user of connection status, data delivery, connection termination, and error conditions. The response primitives are Open Id, Open Failure, Open Success, Deliver, Closing, Terminate, Status Response, and Error. Each is fully defined in the following paragraphs.

6.4.10.1 Open id. This service response primitive informs a ULP of the local connection name assigned by TCP to the connection requested in one of the previous service requests, Unspecified Open, Fully Specified Open, or an Active Open. TCP provides in an Open Id at least the following information:

- a. local connection name
- b. source port
- c. destination port [if known]
- d. destination address [if known]

MIL-STD-1778
12 August 1983

6.4.10.2 Open failure. This service response primitive informs a ULP of the failure of an Active Open service request. TCP provides in an Open Failure at least the following information:

- a. local connection name

6.4.10.3 Open success. This service response primitive informs a ULP of the completion of one of the Open service requests. TCP provides in an Open Success at least the following information:

- a. local connection name

6.4.10.4 Deliver. This service response primitive informs a ULP of the arrival of data across the named connection. TCP provides in a Deliver at least the following information:

- a. local connection name
- b. data
- c. data length
- d. URGENT flag

6.4.10.5 Closing. This service response primitive informs a ULP that the peer ULP has issued a CLOSE service request. Also, TCP has delivered all data sent by the remote ULP. TCP provides in a Closing at least the following information:

- a. local connection name

6.4.10.6 Terminate. This service response primitive informs a ULP that the named connection has been terminated and no longer exists. TCP generates this response as a result of a remote connection reset, service failure, and connection closing by the local ULP. TCP provides in a Terminate at least the following information:

- a. local connection name
- b. description

6.4.10.7 Status response. This service response primitive returns to a ULP the current status information associated with a connection named in a previous Status service request. TCP provides in a Status Response at least the following information:

- a. local connection name
- b. source port
- c. source address
- d. destination port
- e. destination address
- f. connection state
- g. amount of data in octets willing to be accepted by the local TCP
- h. amount of data in octets allowed to send to the remote TCP
- i. amount of data in octets awaiting acknowledgment

MIL-STD-1778
12 August 1983

- j. amount of data in octets pending receipt by the local ULP
- k. urgent state
- l. precedence
- m. security
- n. ULP timeout

6.4.10.8 Error. This service response primitive informs a ULP of illegal service requests relating to the named connection or of errors relating to the environment. TCP provides in an Error response at least the following information:

- a. local connection name
- b. error description

6.5 Extended state machine specification services provided to upper layer. TCP performs in a distributed environment. Hence, an effective model of TCP services can be constructed through the composition of two extended state machines, called local service machines. Figure 5 shows a summary of this "split-state" model. Each local machine is coupled with one ULP of the ULP-pair. Each ULP provides stimuli to its local service machine, in the form of service requests, and receives the resulting reactions, as service responses. Each local machine maintains a complete state record, called a state vector, maintaining a local perspective of the state of the connection. At undetermined intervals, the local machines exchange information (denoted by EXCHANGE), thus modelling communication delay. An extended state machine definition is composed of a machine identifier, a state diagram, a state vector, a set of data structures, an event list, and an events and actions correspondence.

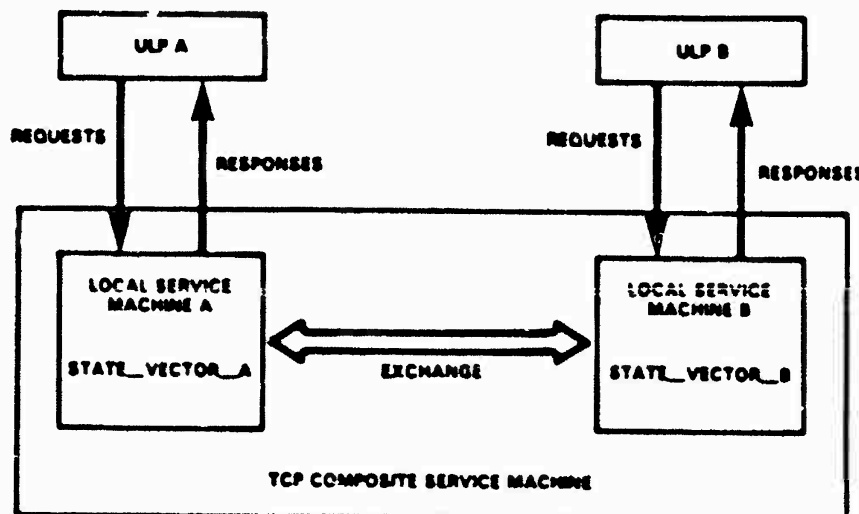


FIGURE 5. Split-state model of TCP services.

MPL-STD-1778
12 August 1983

6.5.1 Machine instantiation identifier. Each local service machine is uniquely identified by the values:

- a. port of ULP A
- b. address of ULP A
- c. port of ULP B
- d. address of ULP B

6.5.1.1 Local connection name. After the first open service request, a TCP uses a shorter name, called a local connection name, to identify a connection in the interactions with its coresident ULP. The Unspecified Passive Open service request does not designate the port and address of the remote ULP and such "half-named" service machines are distinguished by local connection name. A fully-named service machine (if it exists) will be connected to a remote open request rather than a half-named service machine with the same source port and source address. Then, if more than one half-named service machine exists, they are connected to matching fully-named remote open requests at random.

6.5.2 State diagrams. Because of the split-state model presented, both the local service machine state diagram and the composite service machine state diagram are presented. Figure 6 summarizes the service provided by the composite TCP service machine as derived from the composition of two local service machines. The boxes represent the state of the composite service machine; the arrows represent state transitions resulting from the service requests and service responses shown. The "EX" labels represent state changes resulting from the periodic exchanges between local service machines. This diagram serves only as a guide and does not supersede the full definition of the composite service machine in Section 6.5. Abnormal connection termination states are enclosed in the dotted box. These states result from an Abort service request or from TCP service failure.

6.5.2.1 Service state machine defined. Figure 7 summarizes the definition of the service state machine for the local service machine appearing in Paragraph 6.5. This diagram presents the sequence of state changes from the point of view of a single ULP accessing TCP's services. The boxes represent the states of the state machine; the arrows represent state transitions resulting from the service requests and service responses shown. Please note that the diagram is intended only as a summary and does not supersede the formal definition of Paragraph 6.5.

6.5.3 State vector. The service machine vector of a local service machine consists of the following elements:

- a. state - (CLOSED, ACTIVE OPEN, PASSIVE OPEN, ESTABLISHED, CLOSING);
- b. source port - identifier of the local ULP.
- c. source address (sv. source addr) - the internet address naming the location of the local ULP.

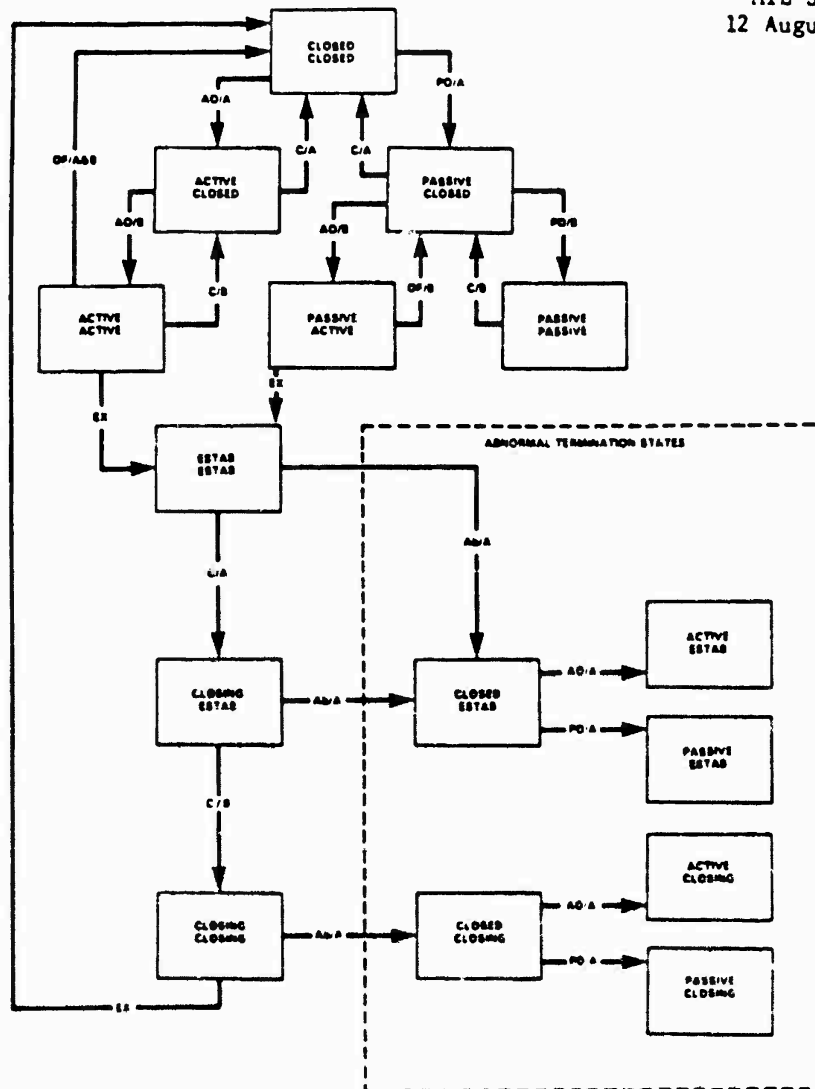
MIL-STD-1778
12 August 1983

FIGURE 6. Composite TCP service state machine diagram.

LEGEND	
ULP Service Requests	TCP Internal Events
PO - passive open, either unspecified or fully specified	EX - exchange of state information between local service entities
AO - active open	T - termination of service due to service failure
S - send	OF - open fail, active open request failed
C - close	
Ab - abort (forces to CLOSED)	

Note: The first "actor" of the ULP-pair is defined to be ULP A, as shown by the notation PO/A, or AO/A.

MIL-STD-1778
12 August 1983

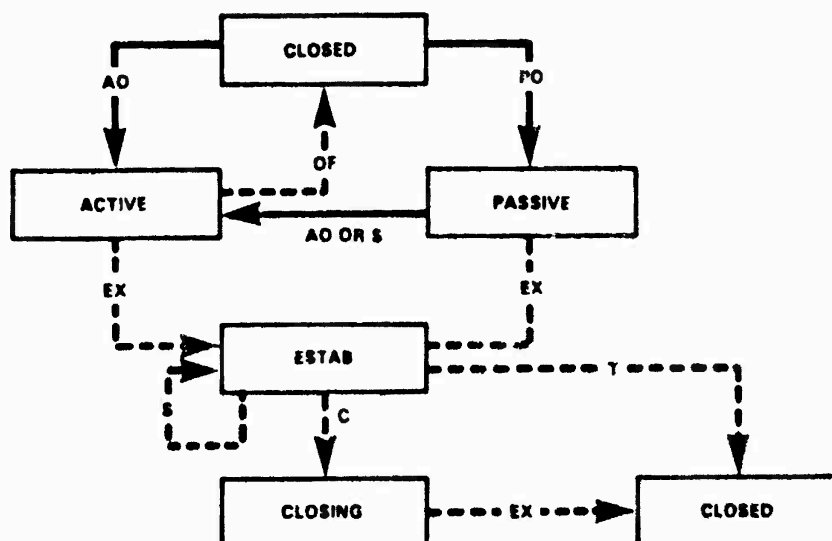


FIGURE 7. TCP local service state machine summary.

TCP LOCAL SERVICE STATE MACHINE SUMMARY

LEGEND	
Service Requests	TCP Service Machine Internal Events
----->>
PO - passive open, either unspecified or fully specified	EX - exchange between service entities
AO - active open, with or without data	T - termination of service due to service failure
S - send	
C - close	
A - abort (always leads to CLOSED state)	

MIL-STD-1778
12 August 1983

- d. destination port - identifier of the remote ULP.
- e. destination address - internet address identifying the location of the remote ULP.
- f. local connection name - the shorthand identifier used in all service responses and service requests except for open requests.
- g. original precedence - precedence level specified by the local ULP in the open request.
- h. actual precedence - precedence level negotiated at connection opening and used during connection lifetime.
- i. security - security information (including security level, compartment, handling restrictions and transmission control code) defined by the local ULP.
- j. sec_ranges - security structure which specifies the allowed ranges in compartment, handling restrictions, transmission control codes and security levels.
- k. ULP timeout - the longest delay allowed for data delivery before automatic connection termination.
- l. ULP timeout_action - in the event of a ULP timeout, determines if the connection is terminated or an error is reported to the ULP.
- m. open mode - the type of open request issued by the local ULP including UNPASSIVE, FULLPASSIVE, and ACTIVE.
- n. send queue - storage location of data sent by the local ULP before transmission to the remote TCP. Each data octet is stored with a timestamp indicating its time of entry.
- o. send queue length - number of entries in the send queue made up of data and timestamp information.
- p. send push - an offset from the front of the send queue indicating the end of push data.
- q. send urgent - an offset from the front of the send queue indicating the end of urgent data.
- r. receive queue - storage location of data received from the remote TCP before delivery to the local ULP.
- s. receive queue length - number of data octets in the receive queue.
- t. receive push - an offset from the front of the receive queue indicating the end of push data.

MIL-STD-1778
12 August 1983

- u. receive urgent - an offset from the front of the receive queue indicating the end of urgent data.
- v. receive allocation - the number of data octets the local ULP is currently willing to receive.

6.5.3.1 Initial state. A state machine's initial state is CLOSED with NULL values for all other state vector elements.

6.5.3.2 Sec range structure. The structure of sec_ranges is largely implementation dependent. In the simplest case, it could be implemented as a quartet. For example:

```
(compartment)(handling restriction)(transmission control code)(sec_level range)
:           :                   :
:           :                   :
:           :                   :
```

In a more complex scenario, the implementation could be tree structured, with the number of branches being ((# compartments) x (# handling restrictions) x (# transmission control codes)). In Figure 8, each branch has its own security_level range.

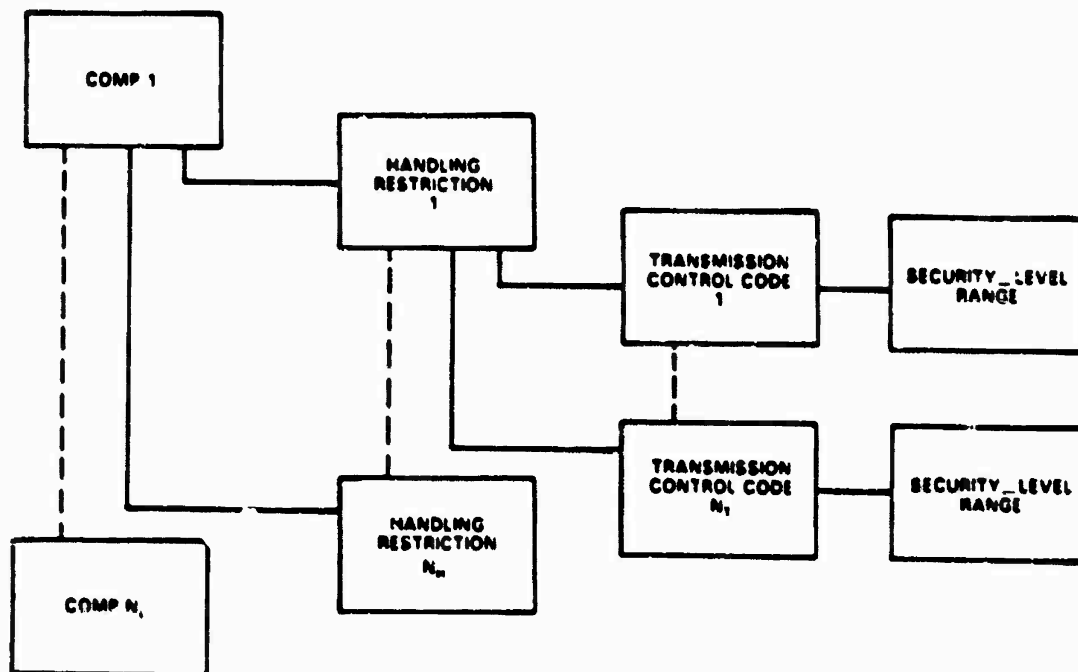


FIGURE 8. Complex sec range structure.

MIL-STD-1778
12 August 1983

6.5.4 Data structures. For clarity in the events and actions section, data structures are declared for the interaction primitives and their parameters. A subset of ADA data constructs, common to most high level languages, is used. However, a data structure may be partially typed or completely untyped where specific formats or data types are implementation dependent.

6.5.4.1 State vector. The definition of the TCP service machine state vector appears in paragraph 6.5.3. The service machine state vectors for the two local TCP service machines are declared as:

```

sv_A : state_vector_type;

sv_B : state_vector_type;

type state_vector_type is
  record
    state : ( CLOSED, ACTIVE_OPEN, PASSIVE_OPEN,
              ESTABLISHED, CLOSING );
    source_addr : address_type;
    source_port : TWO_OCTETS;
    destination_addr : address_type;
    destination_port : TWO_OCTETS;
    lcn : lcn_type;
    sec : security_type;
    sec_ranges : security_structure;
    original_prec : 0..7;
    actual_prec : 0..7;
    ULP_timeout : time_type;
    ULP_timeout_action : integer;
    open_mode : (UNPASSIVE, FULLPASSIVE, ACTIVE);
    send_queue : timed_queue_type;
    send_queue_length : integer;
    send_push : integer;
    send_urg : integer;
    rcv_queue : queue_type;
    rcv_queue_length : integer;
    rcv_push : integer;
    rcv_urg : integer;
    rcv_alloc : integer;
  end record;

type timed_queue_type is queue (1..SIZE_OF_SEND_RESOURCE) of
  record
    data_octet : OCTET;
    timestamp : time_type;
  end record;

type queue_type is queue (1..SIZE_OF_RECV_RESOURCE) of
  data_octet : OCTET;
end record;

```

MIL-STD-1778
12 August 1983

```

type address_type is FOUR_OCTETS;
type lcn_type : undefined; --implementation dependent
type security_struct : undefined; --implementation dependent
type time_type : undefined; --implementation dependent

subtype OCTET is INTEGER range 0..255;
subtype TWO_OCTETS is INTEGER range 0..2**16-1
subtype FOUR_OCTETS is INTEGER range 0..2**32-1;

```

6.5.4.2 From ULP. The from_ULP structure holds the interface parameters and data associated with the service request primitives specified in Section 3.1.1. Although the structure is composed of the parameters from all the service requests, a particular service response will use only those structure elements corresponding to its specified parameters. This structure directly corresponds to the from_ULP structure declared in entity state machine specification, paragraph 9.4.4.2. The from_ULP structure is declared as:

```

type from_ULP_type is
  record
    request_name : (Unspecified_Passive_Open, Full_Passive_Open,
                   Active_Open, Active_Open_with_data,
                   Send, Allocate, Close, Abort, Status);

    source_addr
    source_port
    destination_addr
    destination_port
    lcn
    timeout
    precedence
    security
    sec_ranges
    data
    data_length
    push_flag
    urgent_flag
  end record;

```

6.5.4.3 To ULP. The to_ULP structure holds interface parameters and data associated with the service response primitives, as specified in Section 6.4.10. Although the structure is composed of the parameters from all the service requests, a particular service response will use only those structure elements corresponding to its specified parameters. This structure directly corresponds to the to_ULP structure declared in paragraph 9.4.4.3 of the mechanism specification. The to_ULP structure is declared as:

```

type to_ULP_type is
  record
    service_response : (Open_Id, Open_Fail, Open_Success,
                      Deliver, Status_Response, Terminate,
                      Error);

    source_addr
    source_port

```

MIL-STD-1778
12 August 1983

```

destination_addr
destination_port
len
data
data_length
urgent_flag
error_desc
status_block : status_block_type;
end record;

```

```

type status_block_type is
record
connection_state
send_window
receive_window
amount_of_unacked_data
amount_of_unreceived_data
urgent_state
precedence
security
sec_ranges
timeout
timeout_action
end record;

```

6.5.5 Event list. The events for the TCP service machine are drawn from the service request primitives defined in Section 6.3. Optional service request parameters are shown in brackets. The capitalized list of parameters represent the actual values of the parameters passed by the service primitive. The event list:

- a. Unspecified Passive Open (SOURCE PORT,
[.TIMEOUT] [.TIMEOUT_ACTION]
[.PRECEDENCE] [.SEC_RANGES]);
- b. Full Passive Open (SOURCE PORT,
DESTINATION PORT, DESTINATION ADDRESS,
[.TIMEOUT] T.TIMEOUT_ACTION T.PRECEDENCE]
[.SEC_RANGES]);
- c. Active Open (SOURCE PORT,
DESTINATION PORT, DESTINATION ADDRESS
[.TIMEOUT] T.TIMEOUT_ACTION T.PRECEDENCE]
[.SECURITY]);
- d. Active Open w/data (SOURCE PORT,
DESTINATION PORT, DESTINATION ADDRESS
[.TIMEOUT] T.TIMEOUT_ACTION T.PRECEDENCE]
[.SECURITY]); DATA, DATA_LENGTH, PUSH_FLAG,
URGENT_FLAG);

MIL-STD-1778
12 August 1983

- e. Send (LCN, DATA, DATA LENGTH, PUSH_FLAG, URGENT_FLAG [,TIMEOUT] [,TIMEOUT_ACTION]);
- f. Allocate (LCN, DATA LENGTH)
- g. Close (LCN)
- h. Abort (LCN)
- i. Status (LCN)
- j. NULL - Although no service request is issued by a ULP, certain conditions within the TCP service machine produce a service response.

6.5.6 Events and actions. For the purposes of this definition, the ULP and TCP entities are identified with the capital letters "A" and "B." The first ULP to make a service request is labelled ULP "A"; its local service machine is TCP "A." The peer ULP and its TCP are labelled ULP B and TCP B. The service requests are labelled with the identifier of the issuing ULP, such as Close/A. The service responses are similarly labelled, such as Terminate/B. A service request appearing with a "*" identifier may be issued by either ULP A or ULP B. The appropriate TCP handles the request updating its own state vector if necessary. The service response corresponding to such a request is directed to the appropriate ULP. When a service request is invalid for the current state of the state machine, the service request appears without a parameter list. In this service machine model, "simultaneous" services are treated as unordered sequential events. Hence, CLOSE/A occurring "simultaneously" with CLOSE/B is represented as occurring sequentially without intervening events. The order chosen for the event sequence should not alter the resulting state, so that a sequence such as (CLOSE/A, CLOSE/B) should lead to the same state as the (CLOSE/B, CLOSE/A) sequence. The STATUS event produces the same service response from the TCP service machine in every state. Rather than show these in each state, the STATUS request and STATUS RESPONSE response are shown once here.

Event: STATUS (LCN)

Actions: STATUS_RESPONSE (LCN, SOURCE PORT, SOURCE ADDRESS, DESTINATION PORT, DESTINATION ADDRESS, PRECEDENCE, SECURITY, CONNECTION_STATE, RECEIVE_WINDOW, SEND_WINDOW, AMOUNT_WAITING_ACK, AMOUNT_WAITING_RECEIPT, URGENT_MODE, TIMEOUT, TIMEOUT_ACTION);

6.5.6.1 Event/actions specifications. The following section is organized by composite state. Mirror-image composite states, such as PASSIVE/ACTIVE and ACTIVE/PASSIVE, appear as just one. Only one-way data transfer is represented by the service machine since the data transfer service is symmetric. Thus, a definition of bi-directional data transfer can be provided by duplicating the existing one-way definition. Certain conditions, checks, and

MIL-STD-1778
12 August 1983

groups of actions occur in several places and have been formed into decision functions and action procedures. The decisions function definitions appear in paragraph 6.5.6.2. The action procedure definitions appear in paragraph 6.5.6.3.

6.5.6.1.1 State A = closed, state B = closed.

Event: Unspecified Passive Open/A (SOURCE PORT [,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE] [,SEC_RANGES])

Actions: record_open_parameters (A, UNPASSIVE);
sv_A.lcn := assign_new_lcn;
open_id (sv_A.lcn, sv_A.source_port, sv_A.source_addr, NULL, NULL);
TRANSFER to ULP to the ULP named by sv_A.source_port;
sv_A.state := PASSIVE_OPEN;

Event: Full Passive Open/A (SOURCE PORT, DESTINATION PORT, DESTINATION ADDRESS [,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE] [,SEC_RANGES])

Actions: record_open_parameters (A, FULLPASSIVE);
sv_A.lcn := assign_new_lcn;
open_id (sv_A.lcn, sv_A.source_port, sv_A.source_addr, sv_A.destination_port, sv_A.destination_addr);
TRANSFER to ULP to the ULP named by sv_A.source_port;
sv_A.state := PASSIVE_OPEN;

Event: Active Open/A (SOURCE PORT, DESTINATION PORT, DESTINATION ADDRESS [,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE] [,SEC_RANGES])

Actions: record_open_parameters (A, ACTIVE);
sv_A.lcn := assign_new_lcn;
open_id (sv_A.lcn, sv_A.source_port, sv_A.source_addr, sv_A.destination_port, sv_A.destination_addr);
TRANSFER to ULP to the ULP named by sv_A.source_port;
sv_A.state := ACTIVE_OPEN;

Event: Active Open with data/A (SOURCE PORT, DESTINATION PORT, DESTINATION ADDRESS [,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE] [,SEC_RANGES] DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG)

Actions: sv_A.lcn := assign_new_lcn;
open_id (sv_A.lcn, sv_A.source_port, sv_A.source_addr, sv_A.destination_port, sv_A.destination_addr);
TRANSFER to ULP to the ULP named by sv_A.source_port;
if (room_in(sv_A.send_queue))
then

MIL-STD-1778
12 August 1983

```

        add_to_send_queue(sv_A);
        record_open_parameters(A, ACTIVE);
        sv_A.state := ACTIVE_OPEN;
    else
        openfail (sv_A.lcn);
        TRANSFER to _ULP to the ULP named by sv_A.source_port;

```

Event: Close/A (LCN)
or Abort/A (LCN)
or Allocate/A (LCN, DATA_LENGTH);

Actions: error (sv_A.lcn, "Connection does not exist.");
TRANSFER to _ULP to the ULP named by sv_A.source_port;

6.5.6.1.2 State A = passive open, state B = closed.

Event: Close/A (LCN)
or Abort/A (LCN)

Actions: initialize (sv_A);
sv_A.state := CLOSED;

Event: Unspecified Passive Open/B (SOURCE PORT [,TIMEOUT] [,TIMEOUT_ACTION]
[,PRECEDENCE] [,SEC_RANGES])

Actions: sv_B.lcn := assign_new_lcn;
record_open_parameters (B, UNPASSIVE);
open_id (sv_B.lcn, sv_B.source_port, sv_B.source_addr, NULL, NULL);
TRANSFER to _ULP to the ULP named by sv_B.source_port;
sv_B.state := PASSIVE_OPEN;

Event: Full Passive Open/B (SOURCE PORT,
DESTINATION PORT, DESTINATION ADDRESS
[,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE]
[,SEC_RANGES])

Actions: sv_B.lcn := assign_new_lcn;
record_open_parameters (B, FULLPASSIVE);
open_id (sv_B.lcn, sv_B.source_port, sv_B.source_addr,
sv_B.destination_port, sv_B.destination_addr);
TRANSFER to _ULP to the ULP named by sv_B.source_port;
sv_B.state := PASSIVE_OPEN;

Event: Active Open/B (SOURCE PORT, DESTINATION PORT, DESTINATION ADDRESS
[,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE] [,SECURITY])

MIL-STD-1778
12 August 1983

Actions: record_open_parameters (B, ACTIVE);
sv_B.lcn := assign_new_lcn;
open_id (sv_B.lcn, sv_B.source_port, sv_B.source_addr,
sv_B.destination_port, sv_B.destination_addr);
TRANSFER to ULP to the ULP named by sv_B.source_port;
sv_B.state := ACTIVE_OPEN;

Event: Active Open with data/B (SOURCE_PORT,
DESTINATION_PORT, DESTINATION_ADDRESS
[,TIMEOUT] [,TIMEOUT ACTION] [,PRECEDENCE]
[,SECURITY] DATA, DATA_LENGTH, PUSH_FLAG,
URGENT_FLAG

Actions: sv_B.lcn := assign_new_lcn;
open_id (sv_B.lcn, sv_B.source_port, sv_B.source_addr,
sv_B.destination_port, sv_B.destination_addr);
TRANSFER to ULP to the ULP named by sv_B.source_port;
if (room_in(sv_B.send_queue)
then
add_to_send_queue (sv_B);
record_open_parameters (B, ACTIVE);
sv_B.state := ACTIVE_OPEN;
else
openfail (sv_B.lcn);
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: Allocate/A (LCN, DATA_LENGTH)

Actions: sv_A.recv_alloc := sv_A.recv_alloc + DATA_LENGTH;

Event: Full Passive Open/A ()
or Send/A ()

Actions: error (sv_A.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_A.source_port;

Event: Close/B ()
or Abort/B ()
or Send/B ()
or Allocate/B ()

Actions: error (sv_B.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

MIL-STD-1778
12 August 1983

6.5.6.1.3 State A = active open, state B = closed.

Event: Close/A (LCN)
or Abort/A (LCN)

Actions: initialize (sv A);
sv_A.state := CLOSED;

Event: Allocate/A (LCN, DATA_LENGTH)

Actions: sv_A.recv_alloc := sv_A.recv_alloc + DATA_LENGTH;

Event: Send/A (LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG
[,TIMEOUT] [,TIMEOUT_ACTION])

Actions: if (room in (sv A.send_queue))
then if (TIMEOUT /= NULL)
then sv_A.ulp_timeout := TIMEOUT;
add_to_send_queue (sv A);
else error (sv A.lcn, "Insufficient resources.");
TRANSFER to ULP to the ULP named by sv_A.source_port;

Event: Unspecified Passive Open/B (SOURCE_PORT [,TIMEOUT] [,TIMEOUT_ACTION]
[,PRECEDENCE] [,SEC_RANGES])

Actions: sv_B.lcn := assign_new_lcn;
record_open_parameters (B, UNPASSIVE);
open_id (sv_B.lcn, sv_B.source_port, sv_B.source_addr, NULL, NULL);
TRANSFER to ULP to the ULP named by sv_B.source_port;
sv_B.state := PASSIVE_OPEN;

Event: Full Passive Open/B (SOURCE_PORT,
DESTINATION_PORT, DESTINATION_ADDRESS
[,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE]
[,SEC_RANGES])

Actions: sv_B.lcn := assign_new_lcn;
record_open_parameters (B, FULLPASSIVE);
open_id (sv_B.lcn, sv_B.source_port, sv_B.source_addr,
sv_B.destination_port, sv_B.destination_addr);
TRANSFER to ULP to the ULP named by sv_B.source_port;
sv_B.state := PASSIVE_OPEN;

Event: Active Open/B (SOURCE_PORT,
DESTINATION_PORT, DESTINATION_ADDRESS
[,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE] [,SECURITY])

MIL-STD-1778
12 August 1983

Actions: sv_B.lcn := assign_new_lcn;
record_open_parameters (B, ACTIVE);
open_id (sv_B.lcn, sv_B.source_port, sv_B.source_addr,
sv_B.destination_port, sv_B.destination_addr);
TRANSFER to ULP to the ULP named by sv_B.source_port;
sv_B.state := ACTIVE_OPEN;

Event: Active Open with data/B (SOURCE PORT,
DESTINATION PORT, DESTINATION ADDRESS
[, TIMEOUT] [, TIMEOUT ACTION] [, PRECEDENCE]
[, SECURITY] DATA, DATA_LENGTH, PUSH_FLAG,
URGENT_FLAG)

Actions: sv_B.lcn := assign_new_lcn;
open_id (sv_B.lcn, sv_B.source_port, sv_B.source_addr,
sv_B.destination_port, sv_B.destination_addr);
TRANSFER to ULP to the ULP named by sv_B.source_port;
if (room_in(sv_B.send_queue))

then add_to_send_queue (sv_B);
record_open_parameters (B, ACTIVE);
sv_B.state := ACTIVE_OPEN;

else openfail (sv_B.lcn);
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: Pull Passive Open/A ()
or Active Open/A ()
or Active Open with data/A ()

Actions: error (sv_A.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_A.source_port;

Event: Send/B ()
or Close/B ()
or Abort/B ()

Actions: error (sv_B.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: NULL

Actions: Internal Events

1) if timeout_exceeded (sv_A)
then if (ULP_TIMEOUT_ACTION = 1)

MIL-STD-1778
12 August 1983

```

then openfail (sv A.lcn);
  TRANSFER to ULP to the ULP named by sv_A.source_port;
  initialise (sv_A);
  sv_A.state := CLOSED;
else
  REPORT_TIMEOUT (sv A);

```

6.5.6.1.4 State A = passive open, state B = active open.

Event: Close/*(LCN)
or Abort/*(LCN)

Actions: initialise (sv *);
sv_*.state := CLOSED;

Event: Allocate/*(LCN, DATA_LENGTH)

Actions: sv_*.recv_alloc := sv_*.recv_alloc + DATA_LENGTH;

Event: Send/B (LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG
[,TIMEOUT] [,TIMEOUT_ACTION])

```

Actions: if (room_in(sv_B.send_queue)
then
  add to send_queue(sv_B);
  if (TIMEOUT /= NULL)
  then sv_B.ulp_timeout := TIMEOUT;
else
  error (sv_B.lcn, "Insufficient resources.");
  TRANSFER to ULP to the ULP named by sv_B.source_port;

```

Event: Send/A ()

Actions: error (sv_A.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_A.source_port;

Event: Full Passive Open/*()
or ActiveOpen/*()
or ActiveOpen with data/*()

Actions: error (sv_*.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_*.source_port;

MIL-STD-1778
12 August 1983

Event: NULL

Actions: Internal Events

```

1) if not SEC_RANGE_MATCH (sv_A);
   then
       openfail (sv_B.lcn);
       TRANSFER to ULP to the ULP named by sv_B.source_port;
       initialize (sv_B);
       sv_B.state := CLOSED;

   else --Take greater precedence level to model precedence negotiation;
        --If negotiation is not supported, mismatched precedence
        --is handled the same as mismatched security.
        if (sv_A.original_prec /= sv_B.original_prec)
        then
            sv_A.actual_prec := maximum (sv_A.original_prec,
                                           sv_B.original_prec);
            sv_B.actual_prec := maximum (sv_A.original_prec,
                                           sv_B.original_prec);
            if (sv_A.open_mode = UNPASSIVE)
            then
                sv_A.destination_addr := sv_B.source_addr;
                sv_A.destination_port := sv_B.source_port;
                load_security (sv_A);
                sv_A.state := ESTABLISHED;
                open success (sv_A.lcn);
                TRANSFER to ULP to the ULP named by sv_A.source_port;
                sv_B.state := ESTABLISHED;
                open success (sv_B.lcn);
                TRANSFER to ULP to the ULP named by sv_B.source_port;
                if timeout_exceeded (sv_B);
                then if (ULP_TIMEOUT_ACTION = 1)

OR,
2) if timeout_exceeded(sv_B)
   then
       openfail (sv_B.lcn);
       TRANSFER to ULP to the ULP named by sv_B.lcn;
       initialize (sv_B);
       sv_B.state := CLOSED;
   else
       REPORT_TIMEOUT (sv_B);

6.5.6.1.5 State A = passive open, state B = passive open.

```

Event: Allocate/*(LCN, DATA_LENGTH)

Actions: sv_*.recv_alloc := sv_*.recv_alloc + DATA_LENGTH;

MIL-STD-1778
12 August 1983

Event: Close/*(LCN)
or Abort/*(LCN)

Actions: initialize (sv_*);
sv_*.state := CLOSED;

Event: Full Passive Open/*()
or ActiveOpen/*()
or ActiveOpen with data/*()
or Send/*()

Actions: error(sv_*.lcn, "Illegal request.");
TRANSFER to _ULP to the ULP named by sv_*.source_port;

6.5.6.1.6 State A = active open, state B = active open.

Event: Allocate/*(LCN, DATA_LENGTH)

Actions: sv_*.recv_alloc := sv_*.recv_alloc + DATA_LENGTH;

Event: Close/*(LCN)
or Abort/*(LCN)

Actions: initialize(sv_*);
sv_*.state := CLOSED;

Event: Full Passive Open/*()
or Active Open/*()
or Active Open with data/*()

Actions: error(sv_*.lcn, "Illegal request.");
TRANSFER to _ULP to the ULP named by sv_*.source_port;

Event: Send/*(LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG
[,TIMEOUT] [,TIMEOUT_ACTION])

Actions: if (room_in(sv_*.send_queue)
then
add_to_send_queue(sv_*);
if (TIMEOUT /= NULL)
then sv_*.ulp_timeout := TIMEOUT;
else
error(sv_*.lcn, "Insufficient resources.");
TRANSFER to _ULP to the ULP named by sv_*.source_port;

MIL-STD-1778
12 August 1983

Event: NULL

Actions: Internal Events

```

1) if (sv_A.sec /= sv_B.sec)
    then
        openfail( sv_A.lcn );
        Transfer to ULP to the ULP named by sv_A.source_port;
        openfail( sv_B.lcn );
        Transfer to ULP to the ULP named by sv_B.source_port;
        initialize( sv_A );      sv_A.state := CLOSED;
        initialize( sv_B );      sv_B.state := CLOSED;

    else --take greater precedence level to model precedence negotiation;
        --if negotiation not supported, mismatched precedence
        --is handled just as mismatched security
        if (sv_A.original_prec /= sv_B.original_prec)
            then
                sv_A.actual_prec := maximum(sv_A.original_prec,
                                              sv_B.original_prec);
                sv_B.actual_prec := maximum(sv_A.original_prec,
                                              sv_B.original_prec);

                sv_A.state := ESTABLISHED;
                sv_B.state := ESTABLISHED;
                open_success( sv_A.lcn );
                TRANSFER to ULP to the ULP named by sv_A.source_port;
                open_success( sv_B.lcn );
                TRANSFER to ULP to the ULP named by sv_B.source_port;
            if timeout exceeded (sv_A)
            then if (ULP_timeout_action = 1)

OR,
2) then openfail( sv_A.lcn );
        TRANSFER to ULP to the ULP named by sv_A.source_port;
        initialize(sv_A);
        sv_A.state := CLOSED;

OR,
3) if timeout exceeded (sv_A)
    then if (ULP_timeout_action = 1)
        then
            openfail (sv_B.lcn);
            TRANSFER to ULP to the ULP named by sv_B.source_port;
            initialize (sv_B);
            sv_B.state := CLOSED;
        else
            report_threat;

```

6.5.6.1.7 State A = established, state B = established.

Event: Send/(LCN, DATA, DATA LENGTH, PUSH FLAG, URGENT_FLAG
[,TIMEOUT] [,TIMEOUT_ACTION])

MIL-STD-1778
12 August 1983

```

Actions: if (room_in(sv *.send_queue)
            add_to_send_queue(sv *);
            if (TIMEOUT /= NULL)
            then sv *.ulp_timeout := TIMEOUT;
        else
            error(sv *.lcn, "Insufficient resources.");
            TRANSFER to _ULP to the ULP named by sv *.source_port;

```

Event: Allocate/*(LCN, DATA_LENGTH);

```

Actions: sv *.recv_alloc := sv *.recv_alloc + DATA_LENGTH;
        if (sv *.recv_queue_length > 0)
        then try_to_deliver;

```

Event: Abort/*(LCN)

```

Actions: terminate(sv *.lcn, "User abort.");
        TRANSFER to _ULP to the ULP named by sv *.source_port;
        initialize(sv *);
        sv *.state := CLOSED;

```

Event: Close/*(LCN)

```

Actions: sv *.send_push := sv *.send_queue_length;
        sv *.state := CLOSING;

```

Event: Full Passive Open/*()
or Active Open/*()
or Active Open with data/*()

```

Actions: error(sv *.lcn, "Illegal request.");
        TRANSFER to _ULP to the ULP named by sv *.source_port;

```

Event: NULL

Actions: Internal Events

--For clarity, one-way data transport, from TCP A to TCP B is shown.
--Because the data transport service is symmetric, the following
--text could be duplicated to represent bi-directional data transport.

```

1) if timeout_exceeded(sv_A)
    then if (ULP_timeout_action = 1)
    then
        terminate(sv_A.lcn, "ULP timeout.");
        TRANSFER to _ULP to the ULP named by sv_A.source_port;
        initialize(sv_A);
        sv_A.state := CLOSED;

```

MIL-STD-1778
12 August 1983

```

else
    report_timeout;

OR,
2) if (conditions exist such that no data can be exchanged
    by local state machines )
    then
        terminate(sv A.lcn, "Service failure.");
        TRANSFER to ULP to the ULP named by sv A.source_port;
        terminate(sv B.lcn, "Service failure.");
        TRANSFER to ULP to the ULP named by sv B.source_port;
        initialize(sv A); sv A.state := CLOSED;
        initialize(sv B); sv B.state := CLOSED;

OR,
3) if (the data exchange between local state machines is triggered)
    then
        if (sv A.send_urg /= 0)
            then
                sv B.recv_urg := (sv B.recv_queue_length + sv A.send_urg);

                Dequeue some portion of data equal to "amount"
                (amount may be >= 0) from sv A.send_queue
                and append to sv B.recv_queue;

                if (amount > 0)
                    then
                        sv A.send_queue_length := sv A.send_queue_length - amount;
                        sv B.recv_queue_length := sv B.recv_queue_length + amount;

                        if (sv A.send_urg <= amount)
                            then sv A.send_urg := 0;
                        else sv A.send_urg := sv A.send_urg - amount;

                        if (sv A.send_push <= amount)
                            then sv B.recv_push := sv B.recv_push + sv A.send_push;
                                sv A.send_push := 0;
                            else sv A.send_push := sv A.send_push - amount;
                                try_to_deliver;

```

6.5.6.1.8 State A = established, state B = closing.

Event: Send/A(LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG
[,TIMEOUT] [,TIMEOUT_ACTION])

Actions: if (room_in(sv A.send_queue)
add to send_queue(sv A);
if (TIMEOUT /= NULL)
then sv A.ulp_timeout := TIMEOUT;
else
error(sv A.lcn, "Insufficient resources.");
TRANSFER to ULP to the ULP named by sv A.source_port;

MIL-STD-1778
12 August 1983

Event: Close/A(LCN)

Actions: sv_A.send_push := sv_A.send_queue_length;
sv_A.state := CLOSING;

Event: Allocate/(LCN, DATA_LENGTH);

Actions: sv_*.recv_alloc := sv_*.recv_alloc + DATA_LENGTH;
if (sv_*.recv_queue_length > 0)
then try_to_deliver;

Event: Abort/(LCN)

Actions: initialize(sv_*);
sv_*.state := CLOSED;
terminate(sv_*.lcn, "User abort.");
TRANSFER to ULP to the ULP named by sv_*.source_port;

Event: Send/B()
or Close/B()

Actions: error(sv_B.lcn, "Connection closing.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: Active Open/()
or Active Open with data/()
or Full Passive Open/()

Actions: error(sv_*.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_*.source_port;

Event: NULL

Actions: Internal Events

```
1) if timeout_exceeded(sv_*)  
then if (ULP_timeout_action = 1)  
then  
    terminate(sv_*.lcn, "ULP timeout.");  
    TRANSFER to ULP to the ULP named by sv_*.source_port;  
    initialize(sv_*);  
    sv_*.state := CLOSED;  
else  
    report_timeout (sv_*);
```

MIL-STD-1778
12 August 1983

OR,

```
2) if (conditions exist such that no data can be exchanged
    by local state machines )
    then
        terminate(sv_A.lcn, "Service failure.");
        TRANSFER to ULP to the ULP named by sv_A.source_port;
        terminate(sv_B.lcn, "Service failure.");
        TRANSFER to ULP to the ULP named by sv_B.source_port;
        initialize(sv_A);    sv_A.state := CLOSED;
        initialize(sv_B);    sv_B.state := CLOSED;
```

OR,

```
3) if (contents sv_B.send_queue have all been transferred
    to sv_A.recv_queue and subsequently delivered to ULP A )
    then
        closing( sv_A.lcn );
        TRANSFER to ULP to the ULP named by sv_A.source_port;
```

OR,

```
4) --For clarity, one-way data transport, from TCP A to TCP B is shown.
    --Because the data transport service is symmetric, the following
    --text could be duplicated to represent bi-directional data transport.
    --Note that TCP B is still responsible to reliably transport any
    --data remaining in sv_B.send_queue.
```

```
if (the data exchange between local state machines has been triggered)
then
    if (sv_A.send_urg /= 0)
    then
        sv_B.recv_urg equal := (sv_B.recv_queue_length + sv_A.send_urg);

        Dequeue some portion of data equal to "amount"
        (amount may be >= 0) from sv_A.send_queue
        and append to sv_B.recv_queue;

        if (amount > 0)
        then
            sv_A.send_queue_length := sv_A.send_queue_length - amount;
            sv_B.recv_queue_length := sv_B.recv_queue_length + amount;
            if (sv_A.send_urg <= amount)
            then sv_A.send_urg := 0;
            else sv_A.send_urg := sv_A.send_urg - amount;

            if (sv_A.send_push <= amount)
            then sv_B.recv_push := sv_B.recv_push + amount;
            sv_A.send_push := 0;
            else sv_A.send_push := sv_A.send_push - amount;
            try_to_deliver;
```

MIL-STD-1778
12 August 1983

6.5.6.1.9 State A = closing, state B = closing.

Event: Abort/*(LCN)

Actions: initialize(sv_*);
sv_*.state := CLOSED;
terminate(sv_*.lcn, "User Abort.");
TRANSFER to ULP to the ULP named by sv_*.source_port;

Event: Allocate/*(LCN, DATA_LENGTH);

Actions: sv_*.recv_alloc := sv_*.recv_alloc + DATA_LENGTH;
if (sv_*.recv_queue_length > 0)
then try_to_deliver;

Event: Send/*()
or Close/*()

Actions: error(sv_*.lcn, "Connection closing.");
TRANSFER to ULP to the ULP named by sv_*.source_port;

Event: Active Open/*()
or Active Open with data/*()
or Full Passive Open/*()

Actions: error(sv_*.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_*.source_port;

Event: NULL

Actions: Internal Events

- 1) --For clarity, one-way data transport, from TCP A to TCP B is shown.
--Because the data transport service is symmetric, the following
--text could be duplicated to represent bi-directional data transport.
--Note that TCP B is still responsible to reliably transport any
--data remaining in sv_B.send_queue.

```

if (the data exchange between local state machines has been triggered)
then
    if (sv_A.send_urg != 0)
    then
        sv_B.recv_urg equal := (sv_B.recv_queue_length + sv_A.send_urg);

        Dequeue some portion of data equal to "amount"
        (amount may be >= 0) from sv_A.send_queue
        and append to sv_B.recv_queue;
  
```

MIL-STD-1778
12 August 1983

```

if (amount > 0)
then
    sv_A.send_queue_length := sv_A.send_queue_length - amount;
    sv_B.recv_queue_length := sv_B.recv_queue_length + amount;

    if (sv_A.send_urg <= amount)
    then sv_A.send_urg := 0;
    else sv_A.send_urg := sv_A.send_urg - amount;

    if (sv_A.send_push <= amount)
    then sv_B.recv_push := sv_B.recv_push + amount;
        sv_A.send_push := 0;
    else sv_A.send_push := sv_A.send_push - amount;
    try_to_deliver;

OR,
2) if ((contents sv_B.send_queue have all been transferred
    to sv_A.recv_queue and subsequently delivered to ULP A )
    &
    (contents sv_A.send_queue have all been transferred
    to sv_B.recv_queue and subsequently delivered to ULP B ))
then
    terminate(sv_A.lcn, "Connection closed.");
    TRANSFER to ULP to the ULP named by sv_A.source_port;
    terminate(sv_B.lcn, "Connection closed.");
    TRANSFER to ULP to the ULP named by sv_B.source_port;
    initialize(sv_A);      sv_A.state := CLOSED;
    initialize(sv_B);      sv_B.state := CLOSED;

OR,
3) if timeout_exceeded(sv_*)
then if (ULP_timeout_action = 1)
then
    terminate(sv_*.lcn, "ULP timeout.");
    TRANSFER to ULP to the ULP named by sv_*.source_port;
    initialize(sv_*);
    sv_*.state := CLOSED;
else
    report_timeout (sv_*)

```

--The composite states, CLOSED/ESTABLISHED, CLOSED/CLOSING,
 --ACTIVE/ESTABLISHED, ACTIVE/CLOSING, PASSIVE/ESTABLISHED, AND
 --PASSIVE/CLOSING, are reached after abnormal
 --connection termination caused by either an Abort request or
 --service failure. Because the service request lists for ULP A
 --already appear in other states, these lists are referenced rather
 --than duplicated.

MIL-STD-1778
12 August 1983

6.5.6.1.10 State A = closed, state B = established.

--ULP A's service request list appears in the CLOSED/CLOSED state.

Event: Send/B(LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG
[,TIMEOUT] [,TIMEOUT_ACTION])

Actions: if (room_in(sv_B.send_queue))
 if (TIMEOUT /= NULL)
 then sv_B.ulp_timeout := TIMEOUT;
 add_to_send_queue(sv_B);
 else
 error(sv_B.lcn, "Insufficient resources.");
 TRANSFER to_ULP to the ULP named by sv_B.source_port;

Event: Allocate/B(LCN, DATA_LENGTH);

Actions: sv_B.recv_alloc := sv_B.recv_alloc + DATA_LENGTH;
 if (sv_B.recv_queue_length > 0)
 then try_to_deliver;

Event: Close/B(LCN)

Actions: sv_B.push := sv_B.send_queue_length;
 sv_B.state := CLOSING;

Event: Abort/B(LCN)

Actions: terminate(sv_B.lcn, "User abort.");
 TRANSFER to_ULP to the ULP named by sv_B.source_port;
 initialize(sv_B);
 sv_B.state := CLOSED;

Event: Full Passive Open/B()
 Active Open/B()
 Active Open with Data/B()

Actions: error(sv_B.lcn, "Illegal request.");
 TRANSFER to_ULP to the ULP named by sv_B.source_port;

Event: NULL

Actions: Internal Events

1) terminate(sv_B.lcn, "Remote Abort.");
 TRANSFER to_ULP to the ULP named by sv_B.source_port;
 initialize(sv_B);
 sv_B.state := CLOSED;

MIL-STD-1778
12 August 1983

OR,
2) if timeout exceeded(sv_B)
then if (ULP_timeout_action = 1)
then
 terminate(sv_B.lcn, "User timeout.");
 TRANSFER to ULP to the ULP named by sv_B.source_port;
 initialize(sv_B);
 sv_B.state := CLOSED;
else
 report_timeout (sv_B);

6.5.6.1.11 State A = closed, state B = closing.

--ULP A's service request list appears in the CLOSED/CLOSED state.

Event: Abort/B(LCN)

Actions: terminate(sv_B.lcn, "User Abort.");
 TRANSFER to ULP to the ULP named by sv_B.source_port;
 initialize(sv_B);
 sv_B.state := CLOSED;

Event: Allocate/B(LCN, DATA_LENGTH);

Actions: sv_B.recv_alloc := sv_B.recv_alloc + DATA_LENGTH;
 if (sv_B.recv_queue_length > 0) then try_to_deliver;

Event: Close/B(LCN)
 or Send/B(LCN)

Actions: error(sv_B.lcn, "Connection closing.");
 TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: Full Passive Open/B(LCN)
 Active Open/B(LCN)
 Active Open with Data/B(LCN)

Actions: error(sv_B.lcn, "Illegal request.");
 TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: NULL

Actions: Internal Events

1) terminate(sv_B.lcn, "Remote Abort.");
 TRANSFER to ULP to the ULP named by sv_B.source_port;
 initialize(sv_B);
 sv_B.state := CLOSED;

MIL-STD-1778
12 August 1983

OR,

```
2) if timeout_exceeded(sv_B)
   then if (ULP_timeout_action = 1)
        then
            terminate(sv_B.lcn, "User timeout.");
            TRANSFER to ULP to the ULP named by sv_B.source_port;
            initialise(sv_B);
            sv_B.state := CLOSED;
        else
            report_timeout (sv_B);
```

6.5.6.1.12 State A = active, state B = established.

--ULP A's service request list appears in the ACTIVE/CLOSED state.

Event: Send/B(LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG
[,TIMEOUT] [,TIMEOUT_ACTION])

```
Actions: if (room in(sv_B.send_queue)
            if (TIMEOUT /= NULL)
            then sv_B.ulp_timeout := TIMEOUT;
            add_to_send_queue(sv_B);
        else
            error( sv_B.lcn, "Insufficient resources.");
            TRANSFER to ULP to the ULP named by sv_B.source_port;
```

Event: Close/B(LCN)

```
Actions: sv_B.push := sv_B.send_queue_length;
         sv_B.state := CLOSING;
```

Event: Abort/B(LCN)

```
Actions: terminate(sv_B.lcn, "User abort.");
         TRANSFER to ULP to the ULP named by sv_B.source_port;
         initialise(sv_B);
         sv_B.state := CLOSED;
```

Event: Allocate/B(LCN, DATA_LENGTH);

```
Actions: sv_B.recv_alloc := sv_B.recv_alloc + DATA_LENGTH;
         if (sv_B.recv_queue_length > 0)
             then try_to_deliver;
```

Event: Full Passive Open/B()
Active Open/B()
Active Open with Data/B()

MIL-STD-1778
12 August 1983

Actions: error(sv_B.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: NULL

Actions: Internal Events

1) terminate(sv_B.lcn, "Remote Abort.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;

OR,

2) if timeout exceeded(sv_B)
then if (ULP_timeout_action = 1)
then
terminate(sv_B.lcn, "User timeout.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;
else
report_timeout (sv_B);

OR,

3) if timeout exceeded(sv_A)
then if (ULP_timeout_action = 1)
then
terminate(sv_A.lcn, "User timeout.");
TRANSFER to ULP to the ULP named by sv_A.source_port;
initialize(sv_A);
sv_A.state := CLOSED;
else
report_timeout;

6.5.6.1.13 State A = active, state B = closing.

--ULP A's service request list appears in the ACTIVE/CLOSED state.

Event: Send/B()
or Close/B()

Actions: error(sv_B.lcn, "Connection closing.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: Allocate/B(LCN, DATA_LENGTH);

Actions: sv_B.recv_alloc := sv_B.recv_alloc + DATA_LENGTH;
if (sv_B.recv_queue_length > 0)
then try_to_deliver;

MIL-STD-1778
12 August 1983

Event: Abort/B(LCN)

Actions: terminate(sv_B.lcn, "User abort.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;

Event: Full Passive Open/B()
or Active Open/B()
or Active Open with Data/B()

Actions: error(sv_B.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: NULL

Actions: Internal Events

1) terminate(sv_B.lcn, "Remote Abort.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;

OR,

2) if timeout_exceeded(sv_B)
the if (ULP_timeout_action = 1)
then
terminate(sv_B.lcn, "User timeout.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;
else
report_timeout (sv_B);

OR,

3) if timeout_exceeded(sv_A)
the if (ULP_timeout_action = 1)
then
terminate(sv_A.lcn, "User timeout.");
TRANSFER to ULP to the ULP named by sv_A.source_port;
initialize(sv_A);
sv_A.state := CLOSED;
else
report_timeout (sv_A);

6.5.6.1.14 State A = passive, state B = established.

--ULP A's request list appears in the PASSIVE/CLOSED state.

MIL-STD-1778
12 August 1983

6.5.6.3.1 Add to send queue (sv*). The add to send queue action procedure enqueues the data provided in an Active Open with Data or Send request onto the send_queue of the state vector named by parameter. The data effects of this procedure are:

- Data examined:

from ULP.data	from ULP.urgent_flag
from ULP.data_length	from ULP.push_flag

- Data modified:

sv*.send_queue	sv*.send_push
sv*.send_queue_length	sv*.send_urg

--Add the data, urgent and push information provided by the ULP
--in a SEND to the send_queue of the state vector.

Enqueue contents of from ULP.data to sv*.send_queue, stamping each data octet with the current time;
sv*.send_queue_length := sv*.send_queue_length + from ULP.data_length;

if (from ULP.push_flag = TRUE)
then sv*.send_push := sv*.send_queue_length;

if (from ULP.urgent_flag = TRUE)
then sv*.send_urg := sv*.send_queue_length;

6.5.6.3.2 Assign new lcn. The assign new lcn action procedure assigns a local connection name not currently used for a new open request and subsequent connection. The data effects of this procedure are:

- Data examined: internal resources

- Data modified: none

--The procedure returns the value to be used as the new
--local connection name.

6.5.6.3.3 Error (local connection name, error description). The error action procedure copies the local connection name and error description text supplied by parameter into the to ULP structure. The service response field is assigned to ERROR for subsequent transfer to the ULP. The data effects of the procedure are:

- Data examined: procedure parameters

- Data modified: to ULP.lcn, to ULP.error_desc, to ULP.service_response

to ULP.lcn := local_connection_name;
to ULP.error_desc := error_description;
to ULP.service_response := ERROR;

MIL-STD-1778
12 August 1983

6.5.6.2 Decision functions. The decision functions represent condition checks made in several places in the service state machine definition.

6.5.6.2.1 Room in (state vector name). The room_in decision function compares the amount of space available in the send_queue of the state vector (named by the parameter) against the amount of data provided by the ULP in an Active Open with Data or a Send service request. The data effects of this function are:

- Data examined:

from_ULP.data_length SIZE_OF_SEND_RESOURCES
sv_*.send_queue_length

- Return values:

FALSE - The send_queue cannot accommodate all the data provided in the service request.

TRUE - There is enough room in the send_queue for the data.

```
if (from_ULP.data_length >
    (SIZE_OF_SEND_RESOURCE - sv_*.send_queue_length))
then return (FALSE)
else return (TRUE);
```

6.5.6.2.2 Timeout exceeded (sv*). The timeout_exceeded decision function compares the current time against the age of the data in the send_queue and the specified ULP timeout limit to determine if the ULP timeout has been exceeded. The data effects of this function are:

- Data examined: sv_*.ulp_timeout sv_*.send_queue

- Return values:

FALSE - The data in the send_queue does not exceed the ULP defined timeout limit.

TRUE - Data in the send_queue has exceeded the timeout limit.

--The data at the front of the queue is the oldest.

```
if (sv_*.send_queue_length > 0)
then if (CURRENT_TIME > sv_*.send_queue[0].timeout + sv_*.ulp_timeout)
    then return (TRUE)
    else return (FALSE);
```

6.5.6.3 Action procedures. These routines appear in several places in the service machine definition. The "*" can be replaced by either A or B for delivery to the appropriate ULP.

MIL-STD-1778
12 August 1983

6.5.6.1.15 State A = passive, state B = closing.

--ULP A's request list appears in the PASSIVE/CLOSED state.

Event: Allocate/B(LCN, DATA_LENGTH);

Actions: sv_B.recv_alloc := sv_B.recv_alloc + DATA_LENGTH;
if (sv_B.recv_queue_length > 0)
then try_to_deliver;

Event: Abort/B(LCN)

Actions: terminate(sv_B.lcn, "User abort.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_E);
sv_B.state := CLOSED;

Event: Close/B(LCN)
or Send/B(LCN)

Actions: error(sv_B.lcn, "Connection closing.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: Full Passive Open/B(LCN)
or Active Open/B(LCN)
or Active Open with Data/B(LCN)

Actions: error(sv_B.lcn, "Illegal request.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: NULL

Actions: Internal Events

1) terminate(sv_B.lcn, "Remote Abort.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;

OR,

2) if timeout_exceeded(sv_B)
then if (ULP_timeout_action = 1)
then
terminate(sv_B.lcn, "User timeout.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;
else
report_timeout (sv_B);

MIL-STD-1778
12 August 1983

Event: Send/B(LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG
[,TIMEOUT] [,TIMEOUT_ACTION])

Actions: if (room in(sv_B.send_queue)
if (TIMEOUT /= NULL)
then sv_B.ulp_timeout := TIMEOUT;
add_to_send_queue(sv_B);
else
error(sv_B.lcn, "Insufficient resources.");
TRANSFER to ULP to the ULP named by sv_B.source_port;

Event: Close/B(LCN)

Actions: sv_B.push := sv_B.send_queue_length;
sv_B.state := CLOSING;

Event: Abort/B(LCN)

Actions: terminate(sv_B.lcn, "User abort.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;

Event: Allocate/B(LCN, DATA_LENGTH);

Actions: sv_B.recv_alloc := sv_B.recv_alloc + DATA_LENGTH;
if (sv_B.recv_queue_length > 0)
then try_to_deliver;

Event: NULL

Actions: Internal Events

1) terminate(sv_B.lcn, "Remote Abort.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;

OR,

2) if timeout_exceeded(sv_B)
the if (ULP_timeout_action = 1)
then
terminate(sv_B.lcn, "User timeout.");
TRANSFER to ULP to the ULP named by sv_B.source_port;
initialize(sv_B);
sv_B.state := CLOSED;
else
report_timeout (sv_B);

MIL-STD-1778
12 August 1983

6.5.6.3.4 Load security. The security parameters (including security level, compartment, transmission control code, and handling restrictions) in an incoming segment are loaded into the state vector. The data effects of this function are:

- Data examined:

from_NET.options [security]

- Data modified:

sv.sec

--This would occur after a successful
--sec_range_match.

sv.sec := from_NET.options [security]

6.5.6.3.5 Initialize (sv*). The initialize action procedure clears all values of the state vector named by parameter. The data effects of this procedure are:

- Data examined: procedure parameter

- Data modified: all fields of sv_*

dequeue(sv_*.send_queue,sv_*.send_queue_length);
dequeue(sv_*.rcv_queue,sv_*.rcv_queue_length);
sv_* := null_state_vector;

6.5.6.3.6 Open fail (local connection name). The open_fail action procedure copies the local connection name supplied by parameter and the OPEN_FAIL service response into the to_ULP structure for subsequent transfer to the ULP. The data effects of this procedure are:

- Data examined: procedure parameter

- Data modified: to_ULP.lcn to_ULP.service_response

to_ULP.lcn := local_connection_name;
to_ULP.service_response := OPEN_FAIL;

6.5.6.3.7 Open id (local connection name, source port, source address, destination port, destination addr). The open_id action procedure copies the parameters and the OPEN ID service response into the to_ULP structure for subsequent transfer to the ULP. The data effects of this procedure are:

- Data examined: procedure parameters

MIL-STD-1778
12 August 1983

- Data modified:

to_ULP.lcn	to_ULP.service_response
to_ULP.source_port	to_ULP.destination_port
to_ULP.source_addr	to_ULP.destination_addr

```

to_ULP.lcn := local_connection_name;
to_ULP.source_port := source_port;
to_ULP.source_addr := source_address;
to_ULP.destination_port := destination_port;
to_ULP.destination_addr := destination_addr;
to_ULP.service_response := OPEN_ID;

```

6.5.6.3.8 Open success (local connection name). The open_success action procedure copies the local connection name supplied by parameter, and the OPEN SUCCESS service response into the to_ULP structure for subsequent transfer to the ULP. The data effects of this procedure are:

- Data examined: procedure parameter

- Data modified: to_ULP.lcn to_ULP.service_response

```

to_ULP.lcn := local_connection_name;
to_ULP.service_response := OPEN_SUCCESS;

```

6.5.6.3.9 Report timeout (sv *). The report_timeout action procedure informs the ULP that a ULP_timeout has occurred. The oldest data in the send queue is requeued and the timeout time reset.

The data effects of this function are:

- Data examined:

- Data modified:

```

begin
    error(sv*.lcn,"ULP_timeout")
    transfer to_ULP to the ULP named by sv*_source_port;
    requeue_oldest(sv*);
end;

```

6.5.6.3.10 REQUEUE OLDEST (sv *). The requeue_oldest action procedure removes the oldest data from the send_queue and requeues the data, making it the youngest.

The data effects of this procedure are:

- Data examined:

sv.*send_queue

MIL-STD-1778
12 August 1983

6.5.6.3.11 Terminate (local connection name, description). The terminate action procedure copies the local connection name and description supplied by parameter, and the TERMINATE service response into the to ULP structure for subsequent transfer to the ULP. The data effects of this procedure are:

- Data examined: procedure parameters
 - Data modified:

to_ULP.service_response	to_ULP.lcn
to_ULP.error_desc	
- to_ULP.lcn := local_connection_name;
 to_ULP.error_desc := description;
 to_ULP.service_response := TERMINATE;

6.5.6.3.12 Sec range match? The sec_range_match function checks if the security parameters (including security level, compartment, transmission control code, and handling restrictions) in the incoming segment fit within the security ranges specified in the security list.

The data effects of this function are:

- Data examined only:

from_net.options [security]	sv.sec_ranges
-----------------------------	---------------
- Return values

NO -- The values in the incoming segment are not within the ranges specified in the state vector.

YES -- The values in the incoming segment are within the ranges specified in the state vector.

6.5.6.3.13 Record open parameters (ULP identifier, open mode). The record_open_parameters action procedure copies the values provided by the ULP in an open request to the state vector. The data effects of this procedure are:

- Data examined:

from_ULP.source_port	from_ULP.precedence
from_ULP.source_addr	from_ULP.security
from_ULP.timeout	
- Data modified:

sv_*.source_port	sv_*.original_prec
sv_*.source_addr	sv_*.security
sv_*.destination_port	sv_*.timeout
sv_*.destination_addr	sv_*.open_mode

--Record the socket-pair and connection information
 --provided in the open service request in the state_vector.

MIL-STD-1778
12 August 1983

```

sv_*.port := from ULP.source_port;
sv_*.addr := the address of this TCP;
sv_*.open_mode := open_mode;

--Record timeout, security, and precedence for ULP *
--if provided, otherwise assign default values;

if (from ULP.security /= NULL)
then sv_*.security := from ULP.security
else sv_*.security := DEFAULT_SECURITY;

if (from ULP.sec_ranges /= NULL)
then sv_*.sec_ranges := from ULP.sec_ranges
else sv_*.sec_ranges := DEFAULT_SEC_RANGES;

if (from ULP.precedence /= NULL)
then sv_*.original_prec := from ULP.precedence
else sv_*.original_prec := DEFAULT_PRECEDENCE;

if (from ULP.timeout /= NULL)
then sv_*.ulp_timeout := from ULP.timeout
else sv_*.ulp_timeout := DEFAULT_TIMEOUT;

if (sv_*.open_mode /= UNPASSIVE)
then sv_*.destination_port := from ULP.destination_port;
  sv_*.destination_addr := from ULP.destination_addr;
else sv_*.destination_port := NULL;
  sv_*.destination_addr := NULL;

```

6.5.6.3.14 Try to deliver. The try_to_deliver action procedure determines from the receive allocation, the receive queue size, and the receive push and urgent variables how much data to deliver to the local ULP. This procedure is called from several places for both ULP A and ULP B in the service machine definition. Where the sv_* notation is used, the appropriate state vector name should be replaced. The data effects of this procedure are:

- Data examined: sv_*.source_port
- Data modified:

to_ULP.data	sv_*.recv_push
to_ULP.data_length	sv_*.recv_urg
to_ULP.urgent_flag	sv_*.recv_alloc
to_ULP.lcn	sv_*.recv_queue_length
sv_*.recv_queue	

--The amount of data delivered is based on the amount of pushed
--data waiting and the receive allocation.

```

if (sv_*.recv_push /= 0)
then --As much pushed data allowed by the recv allocation
  --is delivered.

```

MIL-STD-1778
12 August 1983

```
if (sv_*.recv_alloc > sv_*.recv_push)
then
    to_ULP.data_length := sv_*.recv_push;
    sv_*.recv_push := 0;
else
    to_ULP.data_length := sv_*.recv_alloc;
    sv_*.recv_push := sv_*.recv_push - to_ULP.data_length;
else --Without a PUSH, there is no guarantee of delivery.
    --Deliver some amount of data less than or equal to receive
    --allocation (possibly none).

    to_ULP.data_length := some value;

if (to_ULP.data_length /= 0)
then --Update state vector elements and prepare data
    --and parameters for delivery.
    to_ULP.lcn := sv_*.lcn;

    --Urgent data cannot be delivered followed by non-urgent data.
    --If "end-of-urgent" falls in data to be delivered, make
    --two separate deliveries.
    if ((sv_*.recv_urg /= 0) and
        (sv_*.recv_urg < to_ULP.data_length))
    then begin
        --Deliver urgent data alone.
        save := to_ULP.data_length;
        to_ULP.data_length := sv_*.recv_urg;
        sv_*.recv_urg := 0;
        to_ULP.urgent_flag := false;
        Dequeue to_ULP.data_length octets from sv_*.recv_queue
            and place in to_ULP.data;
        sv_*.recv_alloc := sv_*.recv_alloc - to_ULP.data_length;
        sv_*.recv_queue_length := sv_*.recv_queue_length -
            to_ULP.data_length;
        TRANSFER to_ULP to the ULP named by sv_*.source_port;

        --Prepare to deliver remaining non-urgent data.
        to_ULP.data_length := save;
        end;

    --If urgent data follows the data being delivered, inform ULP.
    if (sv_*.recv_urg > to_ULP.data_length)
    then
        to_ULP.urgent_flag := TRUE;
        sv_*.recv_urg := sv_*.recv_urg - to_ULP.data_length;
    else
        to_ULP.urgent_flag := FALSE;
        sv_*.recv_urg := 0;
```

MIL-STD-1778
12 August 1983

```
Dequeue to_ULP.data_length octets from sv_.recv_queue
and place in to_ULP.data;
sv_.recv_alloc := sv_.recv_alloc - to_ULP.data_length;
sv_.recv_queue_length := sv_.recv_queue_length -
                           to_ULP.data_length;
TRANSFER to_ULP to the ULP named by sv_.source_port;
```

MIL-STD-1778
12 August 1983

7. SERVICES REQUIRED FROM LOWER LAYER

7.1 Goal. The goal of this section is to describe the minimum set of services required of the network layer protocol by TCP. The services required are:

- a. data transfer service
- b. generalized network service
- c. error reporting service

7.2 Service descriptions. A description of each service follows.

7.2.1 Data transfer service. The lower layer protocol must provide data transfer between TCP modules in a communication system. Such a system may consist of a single network or a set of interconnected networks forming an internet. Data must arrive at a destination with non-zero probability; some data loss may occur. The data transfer service is not required to preserve the order in which portions of data are supplied by the source upon delivery at the destination. Data delivered is not necessarily error-free. The lower layer protocol must provide data transfer throughout the system. TCP need only supply global addressing and control information with each portion of data to be delivered. Routing and network specific characteristics are handled by the network layer protocol. For example, TCP need not be aware of current topology or packet size restrictions to transmit segments through a particular network.

7.2.2 Generalized network service. The lower layer protocol must provide a means for TCP to select from the transmission service qualities provided by the communication system for each portion of data delivered. The transmission quality parameters must include precedence. Also, the lower layer protocol must provide a means of labelling each portion of data with security information including security level, compartmentation, handling restrictions, and transmission control code (i.e., closed user groups).

7.2.3 Error reporting service. The lower layer protocol must provide error reports to TCP indicating discontinuation of the above services caused by catastrophic conditions in this or lower layer protocols.

MIL-STD-1776
12 August 1983

8. LOWER LAYER SERVICE/INTERFACE SPECIFICATIONS

8.1 Goal. The goal of this section is to specify the minimal subnetwork protocol services required by TCP and the interface through which those services are accessed. The first part defines the interaction primitives and their parameters for the lower interface. The second part contains the abstract machine specification of the lower layer services and interaction discipline.

8.2 Interaction primitives. An interaction primitive defines the purpose and content of information exchanged between two protocol layers. Two kinds of primitives, based on the direction of information flow, are defined. Service requests pass information downward; service responses pass information upward. These primitives need not occur in pairs, nor in a synchronous manner. That is, a request does not necessarily elicit a "response"; a "response" may occur independently of a request. The information associated with an interaction primitive falls into two categories: parameters and data. The parameters describe the data and indicate how the data is to be treated. The data is not examined or modified. The format of interaction primitive information is implementation dependent and so is not specified. A given TCP implementation may have slightly different interfaces imposed by the nature of the network or execution environment. Under such circumstances, the primitives can be modified to include more parameters or additional primitives can be defined. However, all TCPs must provide at least the interface specified below to guarantee that all TCP implementations can support the same protocol hierarchy.

8.2.1 Service request primitives. A single service request primitive is required from the network protocol, NET_SEND.

8.2.1.1 NET_SEND. The NET_SEND primitive contains complete control information for each unit of data to be delivered. TCP passes in a NET_SEND at least the following information:

- a. source address - address of TCP sending data.
- b. destination address - address of the TCP to receive data.
- c. protocol - identifier assigned to recipient TCP.
- d. type of service indicators - relative transmission quality associated with unit of data.
 - precedence - one of eight levels: (P0, P1, P2, P3, P4, P5, P6, P7) where $P0 \leq P1 \leq P2 \leq P3 \leq P4 \leq P5 \leq P6 \leq P7$.
 - reliability one of two levels: (R0, R1) where R0 = normal reliability and R1 = high reliability.
 - delay - one of two levels: (D0, D1) where D0 = normal delay and D1 = low delay.

MIL-STD-1778
12 August 1983

- throughput - one of two levels: (T0, T1) where T0 = normal throughput and T1 = high throughput.
- a. identifier - value distinguishing this portion of data from others sent by this ULP.
- f. don't fragment indicator - flag showing whether the network protocol can fragment data to accomplish delivery.
- g. time to live - value in seconds indicating maximum lifetime of data within the network.
- h. data length - length of data being transmitted.
- i. option data - options requested by TCP from those supported by network protocol including at least security labelling. (The Internet Protocol supports security labelling, source routing, return routing, stream identification, and timestamping.)
- j. data - present when data length is greater than zero.

8.2.2 Service response primitives. A single service response primitive, DELIVER, is required of the network delivery service.

8.2.2.1 NET DELIVER. The NET DELIVER primitive contains the data passed by a source TCP in a NET SEND, along with addressing, quality of service, and option information. TCP receives in a NET_DELIVER at least the following information:

- a. source address - address of sending TCP.
- b. destination address - address of the recipient TCP.
- c. protocol - identifier assigned to TCP as supplied by the sending TCP.
- d. type of service indicators - relative transmission quality associated with unit of data.
 - precedence - one of eight levels: (P0, P1, P2, P3, P4, P5, P6, P7) where P0 <= P1 <= P2 <= P3 <= P4 <= P5 <= P6 <= P7.
 - reliability - one of two levels: (R0, R1) where R0 = normal reliability and R1 = high reliability.
 - delay - one of two levels: (D0, D1) where D0 = normal delay and D1 = low delay.
 - throughput - one of two levels: (T0, T1) where T0 = normal throughput and T1 = high throughput.

MIL-STD-1778
12 August 1983

- e. data length - length of received data (possibly zero).
- f. option data - options requested by source TCP as supported by the network including at least security labelling. (The Internet Protocol supports security labelling, source routing, return routing, stream identification, and timestamping options.)
- g. data - present when data length is greater than zero.

8.2.2.1.1 NET DELIVER error reports. In addition, a NET_DELIVER may contain error reports from the network protocol either together with parameters and data listed above, or, independently of that information. The details of the error reports are network dependent.

8.3 Extended state machine specification of services required from lower layer. The extended state machine in this section defines the behavior of the entire network protocol service machine from the perspective of TCP. This service machine is modelled as a "black box" whose internal actions are hidden from the TCPs using the network protocol's services. The TCP-pair provides stimuli, in the form of service requests, and receives the resulting network protocol reactions, in the form of service responses. An abstract machine definition is composed of a machine identifier, a state diagram, a state vector, a set of data structures, an event list, and an events and actions correspondence.

8.3.1 Machine instantiation identifier. Each upper interface state machine is uniquely identified by the four interaction primitive parameters:

- a. source address
- b. destination address
- c. protocol
- d. identifier

One state machine instance exists for the NET_SEND and NET_DELIVER primitives whose parameters carry the same values.

8.3.2 State diagram. The upper interface state machine has a single state which never changes. No diagram is needed.

8.3.3 State vector. The upper interface state machine has a single state which never changes. No state vector is needed.

8.3.4 Data structures. For clarity in the events and actions section, data structures are declared for the interaction primitives and their parameters. A subset of ADA data constructs, common to most high level languages, is used. However, a data structure may be partially typed or completely untyped where specific formats or data types are implementation dependent.

MIL-STD-1778
12 August 1983

8.3.4.1 To NET. The to NET structure holds the interface parameters and data associated with the NET SEND primitive specified above. This structure directly corresponds to the to NET structure declared in paragraph 9.4.4.4 of the mechanism definition. The to NET structure is declared as:

```
type to NET_type is  
  record  
    source_addr  
    destination_addr  
    protocol  
    type_of_service is  
      record  
        precedence  
        reliability  
        delay  
        throughput  
      end record;  
    identifier  
    time_to_live  
    dont_fragment  
    length  
    data  
    options  
  end record;
```

8.3.4.2 From NET. The from NET structure holds interface parameters and data associated with the NET DELIVER primitive, as specified in paragraph 8.2.2. This structure directly corresponds to the from NET structure declared in paragraph 9.4.4.5 of the mechanism definition. The from NET structure is declared as:

```
type from NET_type is  
  record  
    source_addr  
    destination_addr  
    protocol  
    type_of_service is  
      record  
        precedence  
        reliability  
        delay  
        throughput  
      end record;  
    length  
    data  
    options  
    error  
  end record;
```

MIL-STD-1778
12 August 1983

8.3.5 Event list. The events are drawn from the interaction primitives specified in Section 8.2. An event is composed of a service request primitive and an abstract timestamp to indicate the time of event initiation. The event list is as follows:

- a. NET_SEND(to_NET) at time t.
- b. NULL - Although no service request is issued by TCP, certain conditions within network layer or lower layers produce a service response. These conditions can include duplication of data and subnet errors.

8.3.6 Events and actions. The following section defines the set of possible actions elicited by each event.

8.3.6.1 EVENT = NET SEND (to NET) at time t. Actions:

- a. NET_DELIVER from NET at time t+N to TCP at destination to_NET. destination_addr with all of the following properties:
 - The time elapsed during data transmission satisfies the time-to-live limit, i.e. $N \leq \text{to_NET.time_to_live}$.
 - The quality of data transmission is at least equal to the relative levels specified by to_NET.type_of_service.
 - if (to_NET.dont_fragment = TRUE) then network layer fragmentation has not occurred in transit.
 - if (to_NET.options includes loose source routing) then from_NET.data has visited in transit at least the gateways named by the source provided by NET_SEND.
 - if (to_NET.options includes strict source routing) then from_NET.data has visited in transit only the gateways named by source route provided by NET_SEND.
 - if (to_NET.options includes record routing) then the list of nodes visited in transit is delivered in from_NET.
 - if (to_NET.options includes security labelling) then the security label is delivered in from_NET.
 - if (to_NET.options includes stream identifier) then the stream identifier is delivered in from_NET.
 - if (to_NET.options includes internet timestamp) then the internet timestamp is delivered in from_NET.

MIL-STD-1778
12 August 1983

OR,

- b. NET_DELIVER to TCP source to NET.source_addr indicating one of the following error conditions:
- destination to NET.destination_addr unreachable
 - protocol to NET.protocol unreachable
 - if (to NET.dont_fragment = TRUE) then fragmentation needed but prohibited
 - if (to NET.options contains any option) then parameter problem with option.

OR,

- c. no action

8.3.6.2 EVENT = NULL. Actions:

- a. NET_DELIVER to TCP at source to NET.source_addr indicating the following error condition:
- error conditions in network or lower layers

OR,

- b. NET_DELIVER from NET at time $t+N$ to TCP at destination to NET.destination_addr with all of the following properties:
- The time elapsed during data transmission satisfies the time-to-live limit, i.e. $N \leq \text{from_NET.time_to_live}$.
 - The quality of data transmission is at least equal to the relative levels specified by from_NET.type_of_service.
 - if (from_NET.dont_fragment = TRUE) then network layer fragmentation has not occurred in transit.
 - if (from_NET.options includes loose source routing) then to NET.data has visited in transit at least the gateways named by the source provided by NET_SEND.
 - if (from_NET.options includes strict source routing) then to NET.data has visited in transit only the gateways named by source route provided by NET_SEND.
 - if (from_NET.options includes record routing) then the list of nodes visited in transit is delivered in to NET.
 - if (from_NET.options includes security labelling) then the security label is delivered in to NET.

MIL-STD-1778
12 August 1983

- if (from_NET.options includes stream identifier) then the stream identifier is delivered in to_NET.
- if (from_NET.options includes internet timestamp) then the internet timestamp is delivered in to_NET.

MIL-STD-1778
12 August 1983

9. TCP ENTITY SPECIFICATION

9.1 Goal. The goal of this section is to define the mechanisms of each TCP entity supporting the services provided by the TCP service machine. The first subsection motivates the specific mechanisms chosen and discusses the underlying philosophy of those choices. The second subsection defines the format and use of the TCP segment header fields. The last subsection specifies an extended state machine representation of the TCP entity.

9.2 Overview of TCP mechanisms. The TCP mechanisms are motivated by TCP services, described in Section 5:

- a. multiplexing service
- b. connection management services
- c. data transport service
- d. error reporting service

9.2.1 Service support. Each service could be supported by any of several mechanisms. The selection of mechanisms is guided by design standards including simplicity, generality, flexibility, and efficiency. The mechanism descriptions identify the service or services supported and explain how the mechanisms work. This overview begins with an introduction to some basic terminology used throughout the TCP entity mechanisms discussions. The mechanisms present in a TCP entity are:

- a. flow control windows
- b. duplicate and out-of-order data detection
- c. positive acknowledgments with retransmission
- d. checksum
- e. push
- f. urgent
- g. ULP timeout
- h. ULP timeout action
- i. security and precedence
- j. security ranges
- k. multi-addressing
- l. passive and active open requests
- m. three-way handshake for SYN exchange
- n. open request matching
- o. three-way handshake for FIN exchange
- p. resets

9.2.2 Background and terminology. This section presents the terminology used in the mechanism descriptions. The concept of sequence numbers and sequence space, the variables maintained in a state vector (defined in paragraph 9.4.4.1) and segment header fields (defined in Section 9.3) are introduced. Also presented is a list of the states within the TCP state machine (defined in Section 9.4).

9.2.2.1 Sequence numbers. A fundamental notion in the design of the TCP entity is that every octet of data sent over a connection has a sequence

MIL-STD-1778
12 August 1983

number. These sequence numbers are used by several mechanisms (data ordering, duplicate detection, positive acknowledgment with retransmission, and flow control windows) to provide reliable, ordered data transfer. The sequence number carried in a TCP header is a four octet value designating the sequence number of the first octet of data in the segment. Each successive data octet is numbered sequentially. Thus, each segment is bound to as many consecutive sequence numbers as there are octets of data in the segment.

9.2.2.1.1 Numbering scheme. The numbering scheme is extended to include certain control information as well. This is achieved by implicitly including some control flags in the sequence space so they can be reliably transmitted without confusion (i.e., one and only one copy of the control will be acted upon). Control information is not physically carried in the segment data space. Consequently, we must adopt rules for implicitly assigning sequence numbers to control. The "SYN" and "FIN" flags are the only controls requiring this protection. These controls are used only at connection opening and closing. For sequence number purposes, the "SYN" is considered to occur before the first actual data octet of the segment in which it occurs. When a "SYN" is present then `SEQ.SEQ` is the sequence number of the "SYN." The FIN is considered to occur after the last actual data octet in a segment in which it occurs. The segment length (`LENGTH`) includes both data and sequence space occupying controls. It is essential to remember that the actual sequence number space, ranging from 0 to $2^{32}-1$, is finite though very large. Because the space is finite, all arithmetic dealing with sequence numbers must be performed modulo 2^{32} . This unsigned arithmetic preserves the relationship of sequence numbers as they wrap around from $2^{32}-1$ to 0 again.

9.2.2.2 Connection sequence variables. To maintain a connection, a TCP entity records and updates connection status information in a state vector. (This is also called a Transmission Control Block, or TCB.) Among the status information stored in the state vector are sequence variables describing the data exchange over the connection. A connection carries data in two directions, and so each TCP entity maintains sequence variables for both the data it sends and the data it receives.

9.2.2.2.1 Send variables. Send variables are used to track the status of the send data stream with regard to acknowledgments, urgent data, pushed data, window size and position, and the initial sequence number. This list is a subset of the complete list of all send variables appearing in the state vector definition, paragraph 9.4.3.

- a. `SEND_NEXT` - send next, the sequence number of the next octet of data to be sent.
- b. `SEND_UNA` - send unacknowledged, all octets up to but not including this sequence number have been acknowledged.
- c. `SEND_WNDW` - send window, the number of data octets currently allowed to be sent relative to `SEND_UNA`.
- d. `SEND_URG` - send urgent point, the sequence number of the last octet of urgent data.

MIL-STD-1778
12 August 1983

- e. SEND_PUSH - send push point, the sequence number of the last octet of pushed data.
- f. SEND_LASTUP1 - last window update one, the sequence number carried by the incoming segment used for last window update.
- g. SEND_LASTUP2 - last window update two, the acknowledgment number carried by the incoming segment used for last window update.
- h. SEND_ISN - initial send sequence number, the sequence number of the SYN sent on this connection.

These names correspond to the state vector elements defined in paragraph 9.4.3.

9.2.2.2.2 Send sequence space. If the space of send sequence numbers is pictured as a number line, the following diagram shows the relationships among some of the variables defined above.



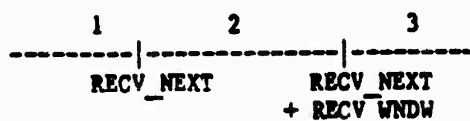
- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers of sent but as yet unacknowledged data
- 3 - sequence numbers allowed for new data transmission
(i.e. the unused send window)
- 4 - future sequence numbers which are not yet allowed

9.2.2.2.3 Receive variables. The receive variables track the receive data stream in regard to acknowledgments, urgent data, pushed data, window size and position, and initial sequence number. These variables are a subset of the state vector elements defined in paragraph 9.4.3.

- a. RECV_NEXT - receive next, the sequence number of the next data octet to be received.
- b. RECV_WNDW - the number of data octets that can currently be received starting from RECV_NEXT.
- c. RECV_URG - receive urgent point, the sequence number of the last octet of urgent data.
- d. RECV_PUSH - receive push point, the sequence number of the last octet of pushed data.
- e. RECV_ISN - initial receive sequence number, the sequence number of the SYN received from the remote TCP.

MIL-STD-1778
12 August 1983

9.2.2.2.4 Receive sequence space. If the space of receive sequence numbers is pictured as a number line, the following diagram shows the relationships among some of the variables defined above.



- 1 - old sequence numbers which have already been accepted
- 2 - sequence numbers allowed to be received
(i.e., the receive window)
- 3 - future sequence numbers not yet allowed

9.2.2.3 Current segment variables. TCP entities communicate in units of exchange called segments. A segment is made up of a header, containing addressing and control information, and a text area, containing a portion of the send or receive data streams. A formal definition of the segment header format appears in Paragraph 9.3. The following header fields and related values are used in the mechanism descriptions.

- a. **SEG.SEQ** - segment sequence number, the sequence number carried in the segment header. It may number the first octet of carried data, number a sequence control flag, or (in an empty segment) indicate the next octet to be sent.
- b. **SEG.ACK** - segment acknowledgment number, the acknowledgment from the sending TCP. That is, the next sequence number expected from the receiving TCP.
- c. **SEG.WNDW** - segment window, the current number of octets that the sending TCP will accept as counted from **SEG.ACK**.
- d. **SEG.URGPTR** - segment urgent pointer, the number of data octets remaining before the end of the urgent data, as counted from **SEG.SEQ**.
- e. **LENGTH** - segment length, number of octets of header and text carried in the segment. This value is supplied as a service response parameter.

9.2.2.4 Connection states. A TCP connection progresses through three phases: opening (or synchronization), maintenance, and closing. The three phases are broken down further into states which represent significant stages in the handshake mechanisms of connection opening and closing. These states correspond to the values assumed by the primary element of the state vector structure, **sv.state**. The TCP entity states are:

MIL-STD-1778
12 August 1983

- a. **LISTEN** - after a passive open request from the local ULP, represents waiting for a connection request from a remote TCP (in the form of a SYN segment).
- b. **SYN_SENT** - after an active open request from the local ULP and having sent an open request (i.e., a SYN), represents waiting for a matching connection open request (i.e., another SYN) from the remote TCP.
- c. **SYN_RECEIVED** - represents waiting for a confirming connection request acknowledgment (i.e., the ACK of the SYN) after having both received and sent connection requests.
- d. **ESTABLISHED** - represents an open connection on which data can be passed between ULPs in both directions.
- e. **FIN_WAIT1** - after a close service request from the local ULP, represents waiting for either a close request (in the form of a FIN segment) from the remote TCP, or an acknowledgment of the close request already sent (i.e., an ACK of the FIN). Data received from remote TCP is delivered to the local ULP.
- f. **FIN_WAIT2** - represents waiting for a connection termination request (i.e., a FIN) from the remote TCP. Data received from remote TCP is delivered to the local ULP.
- g. **CLOSE_WAIT** - represents having received a connection close request (i.e., a FIN) from the remote TCP and waiting for a connection close request from the local ULP. Data sent by the local ULP is sent to the remote TCP.
- h. **LAST_ACK** - represents having both sent and received a connection close request, having acknowledged the remote close request, and waiting for the last acknowledgment from the remote TCP.
- i. **CLOSING** - represents waiting for the acknowledgment of a connection close request (i.e., an ACK of the FIN) from the remote TCP.
- j. **TIME_WAIT** - represents waiting for enough time to pass to ensure the remote TCP has received the acknowledgment of its connection close request.
- k. **CLOSED** - represents no connection.

The full definition of the TCP states, events, and processing appears in Section 9.4.

9.2.3 **Flow control window.** TCP provides a flow control mechanism, called a window, to enable a receiving TCP entity to govern the amount of data transmitted by a sending TCP entity. A window is an "absolute" flow control

MIL-STD-1778
12 August 1983

technique. Absolute flow control defines an interval of sequence numbers corresponding to the amount of data an entity is willing to accept. This technique prevents ambiguity introduced by duplicate segments because permission to transmit is specified as a specific range of sequence numbers rather than an incremental value. The receiving TCP maintains the amount of data allowed for acceptance in the receive variable `RECV_WNDW`. This value is bound to the receive sequence space beginning at `RECV_NEXT`, the next expected data octet. A TCP entity communicates its current receive window to the remote TCP by placing the window in each outbound segment header in the following manner. The window field of the segment header, `SEG_WINDOW`, is a positive integer value expressing the number of acceptable data octets. The acknowledgment number in the segment header, `SEG_ACK`, associates that quantity to the receive sequence space. Thus, the receive window starts with `SEG_ACK` and continues through the number of octets indicated by `SEG_WINDOW`. As each incoming segment is validated by sequence number and acknowledgment (Sections 9.2.3 and 9.2.4), the TCP entity records the window size in the send variable, `SEND_WNDW`.

9.2.3.1 Shrinking windows. A TCP entity is strongly discouraged from "shrinking" its receive window. A window is said to shrink when a TCP entity advertises a large window and subsequently advertises a smaller one without having accepted the difference in data. Such behavior complicates the send data algorithms of the peer entity. For example, a sending TCP may act upon a large window allocation by sending all of the advertised amount. When the window shrinks, data already sent becomes outside the window. The sender must either set back the send variables and remove data from the retransmission queue to "unsend" the data, or else ignore the smaller window. The robustness principle mandates that although a TCP entity does not shrink its own receive window, it will be prepared for such behavior by other entities.

9.2.3.2 Zero windows. Windows can close, that is become zero in length, when a receiving TCP has no more room to receive data, either because the ULP has stopped accepting or because system resources have been temporarily exhausted. In this situation, the sending TCP normally would not send data. And, if no data is generated by the other ULP, the sending entity will receive no new window updates. Without special mechanisms, zero windows could halt data transfer. With a zero send window, a sending TCP must be prepared to accept from the local and send to the remote TCP at least one octet of new data. Also, a sending TCP must transmit segments at regular intervals into the zero window in order to guarantee that the re-opening of the receive window will be reliably reported. The recommended transmission interval in this situation is two minutes. With a zero receive window, a TCP entity receiving a segment with data must still send an acknowledgment showing its next expected sequence number and current window even though it does not accept the data. If the receiving TCP emits an empty ACK segment when opening its receive window, it may resume receiving data more quickly. Even with a zero receive window, a TCP must process the ACK, RST, and URG fields of all acceptable incoming segments.

MIL-STD-1778
12 August 1983

9.2.3.3 Window updates with one-way data flow. In a connection with data flowing primarily in one direction, the window information will be carried in segments marked with the same sequence number. If such segments arrive out-of-order, they cannot be reordered. This situation is not serious, but it does allow the window information to occasionally be based on old reports from the receiver. A strategy to avoid this problem is to check both the sequence number and the acknowledgment number when deciding to update the send window. That is, use the window information from segments carrying either a higher sequence number than previously seen, or the same sequence number and the highest acknowledgment number. The highest sequence number of an incoming segment used for a window update is recorded in the send variable `SEND_LASTUP1`; the highest acknowledgment number in `SEND_LASTUP2`.

9.2.3.4 Window management suggestions. A TCP entity's method of managing its window has significant influence on performance. The following sections discuss certain window management policies and their effects.

9.2.3.4.1 Window size vs. actual capacity. In general, advertised window size is based on the amount of available receive storage. Although indicating large windows encourages transmissions, false window promises can degrade performance. If the window is larger than actual storage capacity, more data may arrive than can be accepted. The excess data is discarded, causing its retransmission, adding unnecessarily to the load on the communications system and the sending TCP.

9.2.3.4.2 Small windows. Allocating very small windows causes data to be transmitted in many small segments. Better performance may be achieved using fewer large segments. In general, if both sending and receiving window management algorithms actively attempt to combine small window allocations into larger windows, the tendency toward small segments can be avoided. One suggestion to avoid small windows is for a receiving TCP to defer updating a window until an allocation is at least X percent of the maximum allocation possible for the connection (where X might be 20 to 40). Thus, the TCP could send an ACK when a segment arrives (without updating the window information), and later send another ACK with the larger window. Another suggestion is for the sending TCP to avoid sending small segments by waiting until the window is "large" before sending data. (Note that acknowledgments should not be delayed or unnecessary retransmissions will result.)

9.2.4 Duplicate and out-of-order data detection. The network protocol layer may duplicate or change the order of segments submitted by TCP for transmission. To compensate, a TCP entity uses sequence numbers to detect out-of-order and duplicate segments. Duplicate segments are discarded. Segments arriving out of order may, depending on implementation choices, be either discarded or saved for subsequent processing. The duplicate detection and sequencing algorithms rely on the unique binding of segment data to sequence space. The algorithms are based on the assumption that all 2^{32} sequence number values are not cycled through before the segment data bound to those sequence numbers has been delivered and acknowledged by the receiver and all duplicate copies of the segments have "drained" from the internet. Without such an assumption, two distinct TCP segments could conceivably be

MIL-STD-1778
12 August 1983

assigned the same or overlapping sequence numbers, causing confusion at the receiver as to which data is new and which is old. A sending TCP entity keeps track of the sequence number of the next data octet to send in the variable SEND_NEXT. In each outgoing segment, the entity records the sequence number of the first data octet in the segment header field, SEG.SEQ, and advances SEND_NEXT by the total amount of data carried in the segment. A receiving TCP entity keeps track of the sequence number of the next data octet expected in the variable RECV_NEXT. That value and the variable RECV_WNDW represent the receive window, or interval of acceptable data octets. This interval is compared against incoming segment sequence numbers to determine their "acceptability."

9.2.4.1 Incoming and unacceptable segments. An incoming segment is defined to be acceptable if any error-free data it carries falls within the receive window. If the segment does not carry data, the segment sequence number must fall within the receive window. When the receive window is zero, a segment is acceptable if its sequence number equals the next expected sequence number, RECV_NEXT. The processing of unacceptable segments is discussed in 9.2.3.

9.2.4.1.1 "In order" data acceptance. The control information, including valid acknowledgment, window and urgent information, must be used from every acceptable segment. However, the policy for taking (i.e., adding to the receive queue) the data of an acceptable segment can be approached in two ways. The first approach takes only in-order data. That is, only data octets with sequence numbers starting at RECV_NEXT and continuing through to either the end of the segment or the end of the receive window (whichever is shorter) are taken. The data octets of acceptable segments with sequence numbers starting beyond RECV_NEXT are not taken. This "in-order" approach allows immediate acknowledgment and delivery to the ULP.

9.2.4.1.2 "In window" data acceptance. The second approach, called "in-window" data acceptance, takes any data falling within the receive window. If the data is not contiguous with previously received data, it is saved for processing until the intervening data arrives. Thus, acknowledgment and delivery will be delayed until a contiguous interval of data arrives.

9.2.5 Positive acknowledgment with retransmission. Another mechanism compensating for network protocol behavior is positive acknowledgment with retransmission. This mechanism replaces data lost or damaged in transit through the use of sequence numbers and acknowledgments. The basic strategy with PAR is for a sending TCP to retransmit a segment at timed intervals until a positive acknowledgment is returned. The mechanism requirements for segment retransmission, acknowledgment acceptance, transmission intervals, and sequence variable manipulation are described below. The PAR strategy requires TCP to keep copies of all segments in order on a "retransmission queue." As each segment is sent, a segment copy is placed on the end of the queue. The retransmission queue holds the data octets whose sequence numbers begin with SEND_UNA, the oldest unacknowledged sequence number, and ends with SEND_NEXT, the next octet to be sent. When all sent data has been acknowledged, SEND_UNA equals SEND_NEXT, and the retransmission queue is empty. When data is placed on the retransmission queue, a timer is set for the interval expected to

MIL-STD-1778
12 August 1983

elapse before its acknowledgment returns. When a segment or an acknowledgment is lost, the retransmission timer will expire and the TCP will retransmit the unacknowledged data. If the original segment was lost or discarded due to damage, the retransmitted segment is accepted as the original at the receiving TCP. If the acknowledgment was lost, the receiving TCP discards the retransmitted segment as a duplicate, but resends its acknowledgment.

9.2.5.1 Acknowledgment generation. Every TCP segment, excluding an initial SYN segment, must carry an acknowledgment indicating current receive variable information. Acknowledgments are carried in the TCP segment header in a four octet field designating the sequence number of the next expected data octet. The acknowledgment mechanism is cumulative so that an ACK of sequence number X indicates that all octets up to but not including X have been received. Thus, a TCP entity sets the ACK field of each outgoing segment to the value of `RCV_NEXT`, implicitly stating that it has successfully received every data octet up to that sequence number. An acknowledgment does not guarantee that data has been delivered to the ULP, but only that the destination TCP has taken the responsibility to do so.

9.2.5.2 ACK validation. Incoming acknowledgments are compared with the send variables to determine their "acceptability." An "acceptable ACK" is one for which the inequality holds: $SEND_UNA \leq SEG_ACK \leq SEND_NEXT$. In other words, the acknowledgment refers to data equal to or beyond that already acknowledged, and yet does not exceed the sequence number of data yet to be sent. If $SEG_ACK < SEND_UNA$, it is an old ACK and is unacceptable. If $SEG_ACK > SEND_NEXT$, it acknowledges data not yet sent, and so is unacceptable. When an acceptable ACK equals `SEND_UNA`, no new data is acknowledged but new window information may be present. When an acceptable ACK is greater than `SEND_UNA`, it becomes the new value for `SEND_UNA`.

9.2.5.3 Retransmission queue removals. Acknowledgments are not only used to update `SEND_LNDW` and `SEND_UNA`, they are also processed with respect to the retransmission queue. When an ACK arrives fully acknowledging a segment on the retransmission queue, the segment copy is removed from the queue. An ACK is said to fully acknowledge a segment copy on the retransmission queue if the sum of the segment copy's sequence number and length is less than or equal to the acknowledgment number of the incoming segment.

9.2.5.4 Retransmission strategies. A TCP implementation may employ one of several retransmission strategies.

- a. **First-only retransmission** - The TCP entity maintains one retransmission timer for the entire queue. When the retransmission timer expires, it sends the segment (or a segment's worth of data) at the front of the retransmission queue and resets the timer.
- b. **Batch retransmission** - The TCP entity maintains one retransmission timer for the entire queue. When the retransmission timer expires, it sends all the segments on the retransmission queue and resets the timer.

MIL-STD-1778
12 August 1983

- c. Individual retransmission - The TCP entity maintains one timer for each segment on the retransmission queue. As the timers expire, the segments are retransmitted individually and their timers reset.

A brief discussion of retransmission strategy trade-offs and their relationship to the acceptance policy appears in Appendix A.

9.2.5.5 Retransmission timeouts. The value of the retransmission timer can have a large effect on the performance of both the connection and the network. A timeout interval that is too short results in unnecessary retransmissions, wasting both TCP processing time and network resources, while one that is too long results in poor throughput and poor response time for the ULP. Ideally, the retransmission interval should equal exactly the time required for a segment to traverse the network to its destination, be processed, and its ACK to traverse the network back to the source. This sum is called the Round Trip Time (RTT) (see Appendix B). Realistically, however, this value is rarely known or constant. Instead, an approximation of this sum can be dynamically computed during the lifetime of a connection.

9.2.6 Checksum. The checksum mechanism supports error-free data transfer service by enabling detection of segments damaged in transit. A checksum value is computed for each outbound segment and placed in the header's checksum field. Similarly, the checksum of each incoming segment is computed and compared against the value of the header's checksum field. If the values do not match, the incoming segment is discarded without being acknowledged. Hence, a damaged segment appears the same as a lost segment and is compensated for by the PAR mechanism. TCP uses a simple one's complement algorithm which covers the segment header, the segment data, and a "pseudo header." The pseudo header is made up of the source address, the destination address, TCP's protocol identifier, and the length of the TCP segment (excluding the pseudo header). By including the extra pseudo header information in the checksum, TCP protects itself from misdelivery by the network protocol. The checksum algorithm is the 16-bit one's complement of the one's complement sum of all 16-bit words in the pseudo header, segment header, and the segment text. If a segment contains an odd number of octets, the last octet is padded on the right with zeros to form a 16-bit word for checksum purposes. While computing the checksum, the checksum field itself is replaced with zeros.

9.2.7 Push. The data that flows on a connection is conceptually a stream of octets. A sending TCP is allowed to collect data from the sending ULP and to segment and send the data at its own convenience. The sending ULP has no way of knowing if the data has been sent or is retained by the local TCP or remote TCP while waiting for a more suitable segment or delivery size. This mechanism enables a ULP to push data through both the local and remote TCP entities. When "push" flag is set in a SEND request, the sending TCP segments and sends all internally stored data within flow control limits. Upon receipt of a pushed segment, the receiving TCP must promptly deliver the pushed data to the receiving ULP. Successive pushes may not be preserved because two or more units of pushed data may be joined into a single pushed unit by either the sending or receiving TCP. Pushes are not visible to the receiving ULP and are not intended to serve as a record boundary marker.

MIL-STD-1778
12 August 1983

9.2.8 Urgent. TCP provides a means to communicate to a receiving ULP that some point in the upcoming data stream has been marked urgent by the sending ULP. Also, the receiving TCP can indicate when all the currently known urgent data has been delivered to the receiving ULP. The objective of the TCP urgent mechanism is to enable the sending ULP to stimulate the receiving ULP to accept some urgent data. TCP does not define what the ULP is required to do with the urgent state information, but the general notion is that the receiving ULP will take action to process the intervening data quickly. The urgent mechanism permits a point in the data stream to be designated as the end of urgent information. Whenever this point is in advance of the variable `RECV_NEXT` at the receiving TCP, that TCP must tell the ULP to go into "urgent mode;" when the receive sequence number catches up to the urgent pointer, the TCP must tell the ULP to go into "normal mode." If the point is updated while the ULP is in urgent mode, the update will be invisible to the ULP. Note that urgent data cannot be delivered together with any non-urgent data that may follow. The mechanism employs an urgent field which is carried in all segments transmitted. The URG control flag indicates that the urgent field is meaningful. The urgent field must be added to the segment sequence number to yield the sequence number of the last octet of urgent data. The absence of this flag indicates that there is no urgent data outstanding. To send an urgent indication the ULP must also send at least one data octet. If the sending ULP also indicates a push, timely delivery of the urgent information to the destination process is enhanced. When an urgent indication appears in a Send service request but the send window does not allow data to be sent immediately, the TCP should send an empty ACK segment with the new urgent information.

9.2.9 ULP timeout and ULP timeout action. The timeout allows a ULP to set up a timeout for all data submitted to the TCP entity. If some data is not successfully delivered to the destination within the timeout period, the state of `ULP_timeout_action` is checked. If `ULP_timeout_action` is 1, the TCP entity will terminate the connection. If it is 0, the TCP entity informs the ULP that a timeout has occurred, and then resets the timer. The timeout appears as an optional parameter in the open request and the send request. Upon receiving either an active open request, or a SYN segment after a passive request, the TCP entity must maintain a timer set for the interval specified by the ULP. As acknowledgments arrive from the remote TCP, the timer is cancelled and set again for the timeout interval. As parameters of the SEND request, timeout and `timeout_action` can change during connection lifetime. If the timeout is reduced below the age of data waiting to be acknowledged, the event dictated by `ULP_timeout_action` will occur. The implementor may choose to allow additional options when informing the ULP in case of a timeout; for example, informing the ULP only on the first timeout.

9.2.10 Security. TCP makes use of the Internet Protocol (IP) options to provide security and precedence on a per connection basis. The security and precedence parameters used in TCP are those defined in IP. Throughout this TCP specification the term "security information" indicates the security parameters used in IP, including security level, compartment, user group, and handling restrictions. In order for a TCP connection to be established,

MIL-STD-1778
12 August 1983

the modules at each end of the connection must agree on the security information and precedence to be associated with the connection. During a passive open, the option exists to pass a security structure of compartments, user groups, and handling restrictions valid for that connection. The implementation of this data type is dependent on local security policy. For each permutation, there exists a security-level range composed of a high and low link. If only one security level is required, the high and low limits would be the same. If no security structure is passed, the implementation dependent default structure is used. When an active open request contains security parameters within the ranges specified by the passive open, a connection is established. Those exact parameters are then used for the duration of the connection.

9.2.11 Precedence level. The precedence level of the connection is negotiated through the exchange of lower bounds by each end during connection opening. The higher of the two values is assigned to the connection. If it is impossible for the end with the lower precedence to raise its level to the higher, or to get the security information to match the connection must be rejected. The inability to match security information or precedence levels is indicated by the receipt of segments after the connection opening with the non-matching information. The connection is then rejected by sending a reset. In addition to sending a reset, the connection attempt with mismatched security information may be reported or recorded in accordance with local standard operating procedures. After the connection is established, the TCP modules must mark outgoing segments with the agreed security information and precedence level. Any incoming segment with security information or precedence level not exactly matching that of the connection causes the termination of the connection. A reset is sent to the remote TCP and the local ULP is informed of the error.

9.2.12 Multiplexing. TCP provides a set of addresses, called port identifiers, to allow for many ULPs within a single host to use TCP communication facilities simultaneously and to identify the separate data streams that a ULP may request. Port identifiers are selected independently by each TCP entity. To provide unique addresses, TCP concatenates an internet address identifying its internet location to a port identifier creating a "socket." Thus, sockets are unique throughout the internetwork and a pair of sockets can uniquely identify each TCP connection. A socket may participate in many connections to different foreign sockets. TCPs are free to associate ports with processes however they choose. However, several basic concepts are necessary in any implementation. There are "well-known" sockets which a TCP entity associates only with the "appropriate" ULP by some means. Well-known sockets are a convenient mechanism for a priori associating a socket address with a standard service. For instance, the "Telnet-Server" process is permanently assigned to a particular socket, and other sockets are reserved for File Transfer, Remote Job Entry, Text Generator, Echoer, and Sink processes (the last three being for test purposes). A socket address might be reserved for access to a "Look-Up" service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification.

MIL-STD-1778
12 August 1983

9.2.13 Connection opening mechanisms. Several mechanisms are used to establish connections between two TCP entities. These mechanisms, including open requests, sequence number synchronization, and initial sequence number generation, are discussed below.

9.2.13.1 Connection open requests. TCP provides a ULP with two ways of opening a connection, called passive open requests and active open requests. The open requests have certain parameters including the local socket and foreign socket naming the connection.

9.2.13.1.1 Passive open request. With a passive open request, the TCP entity assigns a state vector for the connection variables, returns a local connection name, and becomes "receptive" to connections with other ULPs. The foreign socket parameter in a passive open request may be either fully specified or unspecified. That is, when the foreign socket parameter is set to a specific socket value, only the ULP with that socket identifier can be connected. If the foreign socket is unspecified (denoted by all zeros) any ULP can be connected. Such unspecified foreign sockets are allowed only on passive open requests. A service ULP that wished to provide services for unknown other ULPs would issue an unspecified passive open request, supplying its own well-known socket for the local socket.

9.2.13.1.2 Active open request. With an active open request, the TCP entity not only assigns a state vector and a local connection name, but also actively initiates the connection by sending a SYN segment. A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting state vector. The matching of local and foreign sockets determines when a connection has been initiated. There are two principal cases for matching the sockets in the local open requests to the foreign sockets in arriving SYN segments. In the first case, the local open has fully specified the foreign socket so the match must be exact. In the second case, the local passive open has left the foreign socket unspecified so any foreign socket is acceptable as long as the local sockets match. Other possibilities, left up to the implementor, include partially restricted matches. If there are several pending open requests with the same local socket, a foreign active open will be matched to a fully specified open, if one exists, before selecting an unspecified passive open.

9.2.13.2 Three-way handshake. The "three-way handshake" is the mechanism used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCPs simultaneously initiate the procedure. When two ULPs wish to communicate, they issue open requests as described above, instructing their TCPs to initialize and synchronize the mechanism information on each side. However, the potentially unreliable network layer can complicate the process of synchronization. Delayed or duplicate segments from previous connection attempts might be mistaken for new ones. A handshake procedure with clock based sequence numbers is used in connection opening to reduce the possibility of such false connections.

MIL-STD-1778
12 August 1983

9.2.13.2.1 Simplest handshake. In the simplest handshake between an active open request and a passive open request, the TCP pair synchronizes sequence numbers by exchanging three segments. The actively opened TCP entity emits a segment marked with a synchronize control flag, called a "SYN" segment, which is matched at the receiving TCP entity to the passive open request. The receiving TCP entity emits its own SYN also carrying an acknowledgment of the first SYN. That segment is responded to with an acknowledgment. Thus, a three segment exchange establishes the connection. When simultaneous active open requests initiate the connection each TCP receives a SYN segment which carries no acknowledgment after it has sent a SYN. Each respond with an acknowledging segment and a connection is established in four exchanges. Of course, the arrival of an old duplicate SYN segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments will avoid ambiguity in these cases.

9.2.13.2.2 Examples of connection initiations. Several examples of connection initiation follow. Although these examples do not show connection synchronization using data carrying segments, this is perfectly legitimate, so long as the receiving TCP does not deliver the data to the ULP until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the implementation of a trade-off between memory and messages to provide information for this checking. The simplest three-way handshake is shown in the scenario in Section 4. Other examples are shown below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (→) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (←) indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). An "XXX" indicates a segment which is lost or rejected. Comments appear in parentheses. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

9.2.13.2.2.1 Simultaneous connection initiation. Simultaneous initiation is only slightly more complex than a three-way handshake. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED. The principal reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, is used. If the receiving TCP is in a nonsynchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its ULP. This case is discussed under "half-open" connections below.

MIL-STD-1778
12 August 1983

TCP A	TCP B
1. CLOSED	CLOSED
2. SYN-SENT --> <SEQ=100><CTL=SYN>	...
3. SYN-RECEIVED <-- <SEQ=300><CTL=SYN>	<-- SYN-SENT
4. ... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
5. SYN-RECEIVED --> <SEQ=100><ACK=301><CTL=SYN,ACK> ...	
6. ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
7. ... <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED

9.2.13.2.2.2 Old duplicate SYN detection. As a simple example of recovery from old duplicates, consider the following figure. At line 3, an old duplicate SYN arrives at TCP B. TCP B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP B, on receiving the RST, returns to the LISTEN state. When the original SYN finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RSTs sent in both directions.

TCP A	TCP B
1. CLOSED	LISTEN
2. SYN-SENT --> <SEQ=100><CTL=SYN>	...
3. (duplicate) ... <SEQ=90><CTL=SYN>	--> SYN-RECEIVED
4. SYN-SENT <-- <SEQ=300><ACK=91><CTL=SYN,ACK>	<-- SYN-RECEIVED
5. SYN-SENT --> <SEQ=91><CTL=RST>	--> LISTEN
6. ... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
7. SYN-SENT <-- <SEQ=400><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
8. ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>	--> ESTABLISHED

9.2.13.2.2.3 Half-open connections. An established connection is said to be "half-open" if one of the TCPs has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a crash that resulted in loss of memory. Such connections will automatically become reset if an

MIL-STD-1778
12 August 1983

attempt is made to send data in either direction. However, half-open connections are expected to be unusual, and the recovery procedure is somewhat involved. If at site A the connection no longer exists, then an attempt by the ULP at site B to send any data on it will result in the site B TCP receiving a reset control message. Such a message indicates to the site B TCP that something is wrong, and it is expected to abort the connection. Assume that two ULPs A and B are communicating with one another when a crash occurs causing loss of memory to A's TCP. Depending on the operating system supporting ULP A's TCP, it is likely that some error recovery mechanism exists. When the TCP is up again, ULP A is likely to start again from the beginning or from a recovery point. As a result, ULP A will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case, it receives the error message "connection not open" from the local (ULP A's) TCP. In an attempt to establish the connection, ULP A's TCP will send a segment containing SYN. This scenario leads to the example shown in figure 10. After TCP A crashes, the ULP attempts to open the connection again. TCP B, in the meantime, thinks the connection is open. When the SYN arrives at line 3, TCP B, being in a

synchronized state, sees the incoming segment outside the window and responds with an acknowledgment indicating what sequence it next expects to hear (ACK 100). TCP A sees that this segment does not acknowledge anything it sent and, being unsynchronized, sends a reset (RST) because it has detected a half-open connection. TCP B aborts at line 5. TCP A will continue to try to establish the connection; the problem is now reduced to the basic three-way handshake.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. CLOSED	ESTABLISHED
3. SYN-SENT --> <SEQ=400><CTL=SYN>	--> (??)
4. (!!) <-- <SEQ=300><ACK=100><CTL=ACK>	<-- ESTABLISHED
5. SYN-SENT --> <SEQ=100><CTL=RST>	--> (Abort!!)
6. SYN-SENT	CLOSED
7. SYN-SENT --> <SEQ=400><CTL=SYN>	-->

9.2.13.2.2.4 Alternate case 1. An interesting alternative case occurs when TCP A crashes and TCP B tries to send data on what it thinks is a synchronized connection. This is illustrated in the next figure. In this case, the data arriving at TCP A from TCP B (line 2) is unacceptable because no such connection exists, so TCP A sends a RST. The RST is acceptable so TCP B processes it and aborts the connection.

MIL-STD-1778
12 August 1983

- | TCP A | TCP B |
|--|-------------------------|
| 1. (CRASH) | (send 300, receive 100) |
| 2. (??) <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK> <-- ESTABLISHED | |
| 3. --> <SEQ=100><CTL=RST> | --> (ABORT!!) |

9.2.13.2.2.5 Alternate case 2. In the following figure, TCPs A and B with passive opens are waiting for SYNs. An old duplicate arriving at TCP B (line 2) stirs B into action. A SYN-ACK is returned (line 3) and causes TCP A to generate a RST (the ACK in line 3 is not acceptable). TCP B accepts the reset and returns to its passive LISTEN state.

- | TCP A | TCP B |
|---|-------------------------|
| 1. LISTEN | LISTEN |
| 2. ... <SEQ=Z><CTL=SYN> | --> SYN-RECEIVED |
| 3. (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK> | <-- SYN-RECEIVED |
| 4. --> <SEQ=Z+1><CTL=RST> | --> (return to LISTEN!) |
| 5. LISTEN | LISTEN |

A variety of other cases is possible, all of which are accounted for by the reset generation and processing.

9.2.13.3 Initial sequence number selection. TCP imposes no restrictions on a particular connection being used over and over again. A connection is only named by a pair of sockets. New instances of a connection will be referred to as incarnations of the connection. The problem that arises is how to identify duplicate segments from previous incarnations of the connection. This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished. To avoid confusion, segments from one incarnation of a connection must not be used while the same sequence numbers may still be present in the network from an earlier incarnation. This must be assured, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using. Thus, a clock-based initial sequence number generation procedure has been defined.

9.2.13.4 ISN generator. When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32-bit ISN. The generator is bound to a (possibly fictitious) 32-bit clock whose low order bit is incremented roughly every 4 microseconds. Thus, the ISN cycles approximately every 4.55 hours. Assuming segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours, ISNs will be unique.

MIL-STD-1778
12 August 1983

9.2.14 Connection closing synchronization. Connection closing is handled similarly to connection establishment. The following mechanism, including close request and fin exchange, support the reliable data transport and graceful connection closing services.

9.2.14.1 Close requests. A close request indicates that the local ULP has completed its data transfer over the connection. A ULP may close a connection at any time on its own initiative. Closing connections is intended to be a graceful operation in the sense that outstanding send requests will be transmitted (and retransmitted), as flow control permits, until all have been serviced. Thus, it should be acceptable to make several send requests, followed by a close request, and expect all the data to be sent to the destination ULP. It should also be clear that ULPs should continue to accept data on closing connections, since the other ULP may be trying to transmit the last of its data. Thus, a close request means "I have no more to send" but does not mean "I will receive no more." It may happen (if the upper level protocol is not well thought out) that the closing side is unable to get rid of all its data before timing out. In this event, a close turns into abort request, and the closing TCP gives up. Because closing a connection requires communication with the foreign TCP, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP replies to the close request will result in error responses. A close service request also implies the push function.

9.2.14.2 FIN exchange examples. The FIN control flag in the segment header is exchanged with the same synchronization mechanism, the three-way handshake, used for connection opening. From the TCP entity perspective, there are essentially three cases for FIN exchange. One, the local ULP initiates connection closing with a CLOSE service request. Two, the remote TCP entity sends a FIN segment indicating that the remote ULP has issued a close request. Three, both ULPs simultaneously issue close requests.

9.2.14.2.1 Case 1: local ULP initiates connection close. In this case, a FIN segment can be constructed and placed on the outgoing segment queue. No further send requests from the ULP will be accepted by the TCP, and it enters the FIN-WAIT-1 state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP has both acknowledged the FIN and sent a FIN of its own, the first TCP can ACK this FIN. Note that a TCP receiving a FIN will ACK but not send its own FIN until its ULP has closed the connection also.

TCP A	TCP B
1. ESTABLISHED	ESTABLISHED
2. (Close) FIN-WAIT-1 --> <SEQ=100><ACK=300><CTL=FIN,ACK>	--> CLOSE-WAIT
3. FIN-WAIT-2 <-- <SEQ=300><ACK=101><CTL=ACK>	<-- CLOSE-WAIT
4. TIME-WAIT <-- <SEQ=300><ACK=101><CTL=FIN,ACK>	(Close) <-- LAST-ACK

MIL-STD-1778
12 August 1983

5. TIME-WAIT --> <SEQ=101><ACK=301><CTL=ACK> --> CLOSED
6. (2 MSL)
CLOSED

9.2.14.2.2 Case 2: TCP receives FIN from remote TCP. If an unsolicited FIN arrives from the network, the receiving TCP can ACK it and tell the ULP that the connection is closing. The ULP will respond with a close request, upon which the TCP can send a FIN to the other TCP after sending any remaining data. The TCP then waits until its own FIN is acknowledged whereupon it deletes the connection. If an ACK is not forthcoming, after the ULP timeout the connection is aborted and the ULP is informed.

9.2.14.2.3 Case 3: ULPs close simultaneously. Simultaneous close requests by both ULPs at each end of a connection cause FIN segments to be exchanged. When all segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

TCP A	TCP B
1. ESTABLISHED	ESTABLISHED
2. (ULP A issues CLOSE) FIN-WAIT-1 --> <SEQ=100><ACK=300><CTL=FIN,ACK> <-- <SEQ=300><ACK=100><CTL=FIN,ACK> <-- ... <SEQ=100><ACK=300><CTL=FIN,ACK> -->	(ULP B issues CLOSE) ... FIN-WAIT-1 <-- <SEQ=300><ACK=100><CTL=FIN,ACK> <-- ... <SEQ=100><ACK=300><CTL=FIN,ACK> -->
3. CLOSING --> <SEQ=101><ACK=301><CTL=ACK> <-- <SEQ=301><ACK=101><CTL=ACK> <-- ... <SEQ=101><ACK=301><CTL=ACK> -->	... CLOSING <-- <SEQ=301><ACK=101><CTL=ACK> <-- ... <SEQ=101><ACK=301><CTL=ACK> -->
4. TIME-WAIT (2 MSL) CLOSED	TIME-WAIT (2 MSL) CLOSED

9.2.14.3 Quiet time concept. While the clock-based ISN generation prevents overlap of sequence number use under normal conditions, special measures must be taken in situations where a host crashes (or restarts), resulting in a TCP's loss of knowledge concerning the sequence numbers in use on active connections, and the current ISN value. After crash recovery, a TCP may create segments containing the same or overlapping sequence numbers as those in precrash connection incarnations, causing confusion and misdelivery at the receiver. Even hosts managing to remember the time of day used as a basis for ISN selection are not immune to this problem, as the following example illustrates:

"Suppose, for example, that a connection is opened starting with sequence number S. Suppose that this connection is not heavily used and that eventually the initial sequence number function (ISN(t)) takes on a

MIL-STD-1778
12 August 1983

value equal to the sequence number, say S_1 , of the last segment sent by this TCP on a particular connection. Now suppose, at this instant, the host crashes, recovers, and establishes a new incarnation of the connection. The initial sequence number chosen is $S_1 = ISN(t)$ -- last used sequence number on the old incarnation of the connection! If the recover occurs quickly enough, any old duplicates in the network bearing sequence numbers in the neighborhood of S_1 may arrive and be accepted as new packets by the receiver of the new incarnation of the connection."

The problem is that the recovering host may not know for how long it crashed, nor does it know whether there are still old duplicates in the system from earlier connection incarnations.

9.2.14.3.1 "Keep quiet" concept. One way to handle these situations is to require that a TCP must "keep quiet", that is, refrain from emitting segments, for a maximum segment lifetime (MSL) before assigning any sequence numbers. This quiet time restriction allows the segments from earlier connection incarnations to drain from the network.

For this specification, the MSL is assumed to be 2 minutes. This is an engineering choice, and may be changed as experience dictates. TCP implementors violating this restriction run the risk of causing some old data to be accepted as new or new data rejected as old duplicates. Note that if a TCP is reinitialized yet retains its knowledge of sequence numbers in use, the quiet time restriction does not apply; however, care should be taken to use sequence numbers larger than those recently used.

9.2.15 Resets. One of the control flags of the TCP header is the reset flag. A segment carrying a reset flag set true is called a reset. Resets are used to abruptly close established connections, refuse connection attempts, and respond to segments apparently not intended for the current incarnation of a connection. The following paragraphs define the rules for reset generation and for reset validation and processing.

9.2.15.1 Reset generation. Each paragraph below specifies when a reset should be sent, the sequence number and, when needed, the acknowledgment number necessary to make the reset segment acceptable to the remote TCP. When either ULP of the communicating ULP-pair issues an Abort service request, its local TCP informs the remote TCP with a reset segment carrying a sequence number field equal to `SEND_NEXT`. As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case. Specific examples of reset generation in response to misdirected segments are presented in three groups of states:

9.2.15.1.1 When connection does not exist. When the connection does not exist (i.e., its state is `CLOSED`) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to nonexistent connections are rejected in this manner. If such an incoming

MIL-STD-1778
12 August 1983

segment has an ACK field, the reset segment takes its sequence number from the ACK field of the incoming segment; otherwise, the reset segment takes a sequence number value of zero and an acknowledgment number equal to the sum of the sequence number and text length of the incoming segment. The connection remains in the CLOSED state.

9.2.15.1.2 When connection is in any nonsynchronized state. When the connection is in any nonsynchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), a reset is sent in the following cases: The incoming segment acknowledges something not yet sent (that is, the segment carries an unacceptable ACK), or an incoming segment carries security information which does not exactly match that designated for the connection. Resets generated in the nonsynchronized states are made acceptable as follows. When the incoming segment has an ACK field, the reset segment takes its sequence number from the ACK field of the incoming segment; otherwise, the reset segment carries a sequence number equal to zero and acknowledgment field set to the sum of the sequence number and text length of the incoming segment. The connection remains in the same state.

9.2.15.1.3 When connection is in a synchronized state. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT1, FIN-WAIT2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (such as one with an out-of-window sequence number or an unacceptable acknowledgment number) must elicit only an empty acknowledgment segment containing the current send sequence number (SEND NEXT) and an acknowledgment indicating the next sequence number expected to be received (RECV NEXT). (Note that if the unacceptable segment is an empty ACK segment, replying with an ACK may result in a cascade of ACKs. In general, do not ACK an unacceptable empty ACK segment.) The connection remains in the same state. If an incoming segment has security information or a precedence level which does not exactly match those designated for the connection, a reset is sent; the connection enters the CLOSED state. The reset segment takes its sequence number from the ACK field of the incoming segment.

9.2.15.2 Reset processing. In all states except SYN-SENT, all reset (RST) segments are validated by checking their sequence number fields. A reset is valid if its sequence number is in the connection's receive window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is valid if the ACK field acknowledges the SYN. The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state; otherwise, the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the ULP and goes to the CLOSED state.

9.3 TCP header format. A summary of the contents of a TCP header follows: Note that each tick mark represents one bit position. Each field description below includes its name, an abbreviation, and the field size. Where applicable, the units, the legal range of values, and a default value appears.

MIL-STD-1778
12 August 1983

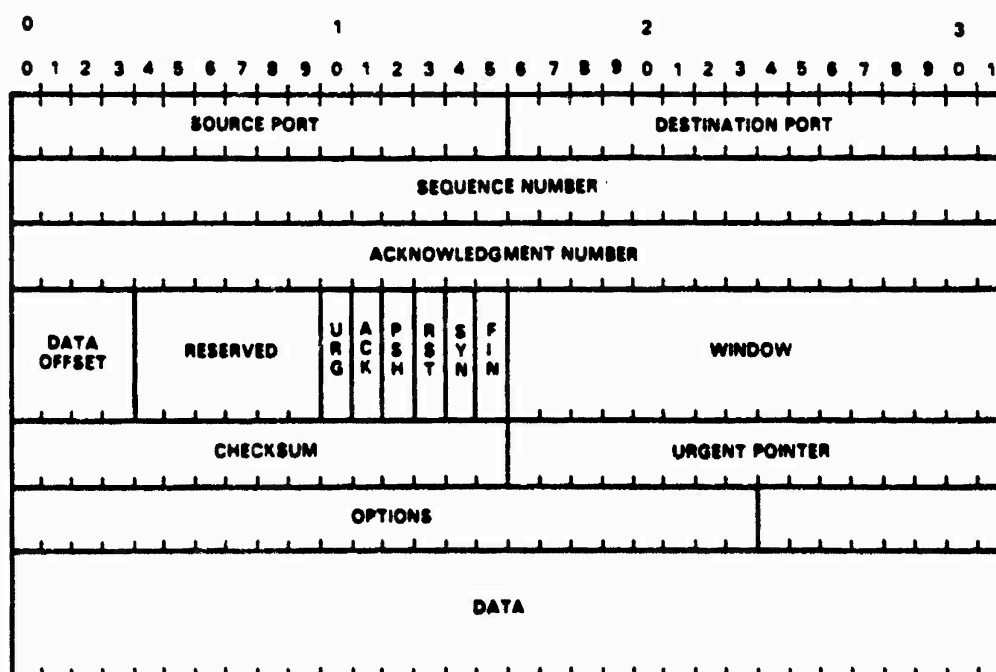


FIGURE 9. TCP header format.

9.3.1 Source port.

abbrev: SRC PORT field size: 16 bits

The source port number.

9.3.2 Destination port.

abbrev: DEST PORT field size: 16 bits

The destination port number.

9.3.3 Sequence number.

abbrev: SEQ field size: 32 bits
units : octets range: 0 - $2^{32}-1$

Usually, this value represents the sequence number of the first data octet of a segment. However, if a SYN is present, the sequence number is the initial sequence number (ISN) covering the SYN; the first data octet is then numbered ISN+1.

MIL-STD-1778
12 August 1983

9.3.4 Acknowledgment number.

abbrev: ACK field size: 32 bits
units: octets range: 0 - 2**32-1

If the ACK control bit is set, this field contains the value of the next sequence number that the sender of the segment is expecting to receive.

9.3.5 Data offset.

abbrev: none field size: 4 bits default: 5
units: 32-bits range: 5 - 15

This field indicates the number of 32 bit words in the TCP header. From this value, the beginning of the data can be computed. The TCP header (even one including options) is an integral number of 32 bits long.

9.3.6 Reserved.

abbrev: none field size: 6 bits

Reserved for future use. Must be set to zero.

9.3.7 Control flags.

abbrev: below field size: 6 bits(from left to right)

URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

These flags carry control information used for connection establishment, connection termination, and connection maintenance.

9.3.8 Window.

abbrev: WNDW field size: 2 octets
units: octets range: 0 - 2**16-1 default: none

The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

9.3.9 Checksum.

abbrev: none field size: 2 octets

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. The checksum also covers a 96 bit pseudo header conceptually prefixed to the TCP header. This pseudo

MIL-STD-1778
12 August 1983

header contains the Source Address, the Destination Address, the Protocol, and TCP segment length. The checksum algorithm is defined in paragraph 9.2.6.

9.3.10 Urgent pointer.

abbrev: URGPTR field size: 2 octets
units: octets range: 0 - 2¹⁶-1 default: 0

This field indicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only to be interpreted in segments with the URG control bit set.

9.3.11 Options.

abbrev: OPT field size: variable

If present, options occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

- a. A single octet of option-kind.
- b. An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets. Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

9.3.11.1 Specific option definitions.

9.3.11.1.1 End of option list.

00000000
KIND = 0

FIGURE 10. End of option list code.

MIL-STD-1778
12 August 1983

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

9.3.11.1.2 No-operation.

00000001

KIND = 1

FIGURE 11. No-operation option code.

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

9.3.11.1.3 Maximum segment size.

00000010	00000100	MAX SEG SIZE
----------	----------	--------------

KIND = 2 LENGTH = 4

FIGURE 12. Maximum segment size option.

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

9.3.12 Padding.

abbrev: none field size: variable

The padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

9.4 Extended state machine specification of TCP entity. The TCP protocol entity is specified with an extended state machine made up of a set of states, a set of transitions between states, and a set of input events causing the state transitions. The following specification is made up of a machine instantiation identifier, a state diagram, a state vector, data structures, an event list, and a correspondence between events and actions. In addition, an extended state machine has an initial state whose value is assumed at state machine instantiation.

MIL-STD-1776
12 August 1980

9.4.1 Machine instantiation identifier. One state machine instance exists for each connection. A connection, and hence a state machine, is uniquely named by either of the two machine instantiation identifiers that exist: the socket pair and the local connection name.

9.4.1.1 Socket pair identifier. TCP segments delivered by the network and connection establishment service requests (Active Open, Active Open with Data, Full Passive Open, and Unspecified Passive Open) carry and thus are bound to a connection with the following values:

- a. source address
- b. source port
- c. destination address
- d. destination port

9.4.1.2 Local connection name. A TCP entity assigns an identifier, a local connection name, that appears in all service responses and all service requests except for active and passive open requests.

9.4.2 State diagram. The following diagram summarizes the state machine for the TCP entity.

Please note the diagram is intended only as a summary and does not supersede the formal definition that follows.

9.4.3 State vector. The elements comprising the state vector of a TCP entity appear below. Each element name is followed by the name of the corresponding record element in the state vector structure "sv" declared in Section 6.3.4.1.

- a. state name (sv.state): the current state of the entity state machine from the following list: CLOSED, LISTEN, SYN_RECVD, SYN_SENT, ESTAB, FIN_WAIT1, FIN_WAIT2, CLOSE_WAIT, CLOSING, LAST_ACK, TIME_WAIT.
- b. source address (sv.source_addr): the internet address naming the location of the local ULP.
- c. source port (sv.source_port): the identifier of the local ULP.
- d. destination address (sv.destination_addr): the internet address of the location of the the ULP at the other end of the connection.
- e. destination port (sv.destination_port): the identifier of the ULP at the other end of the connection.
- f. lcn (sv.lcn): local connection name, the identifier associated with this end of the connection.
- g. open mode (sv.open_mode): the type of open request issued by the local ULP, either ACTIVE or PASSIVE.

MIL-STD-1778
12 August 1983

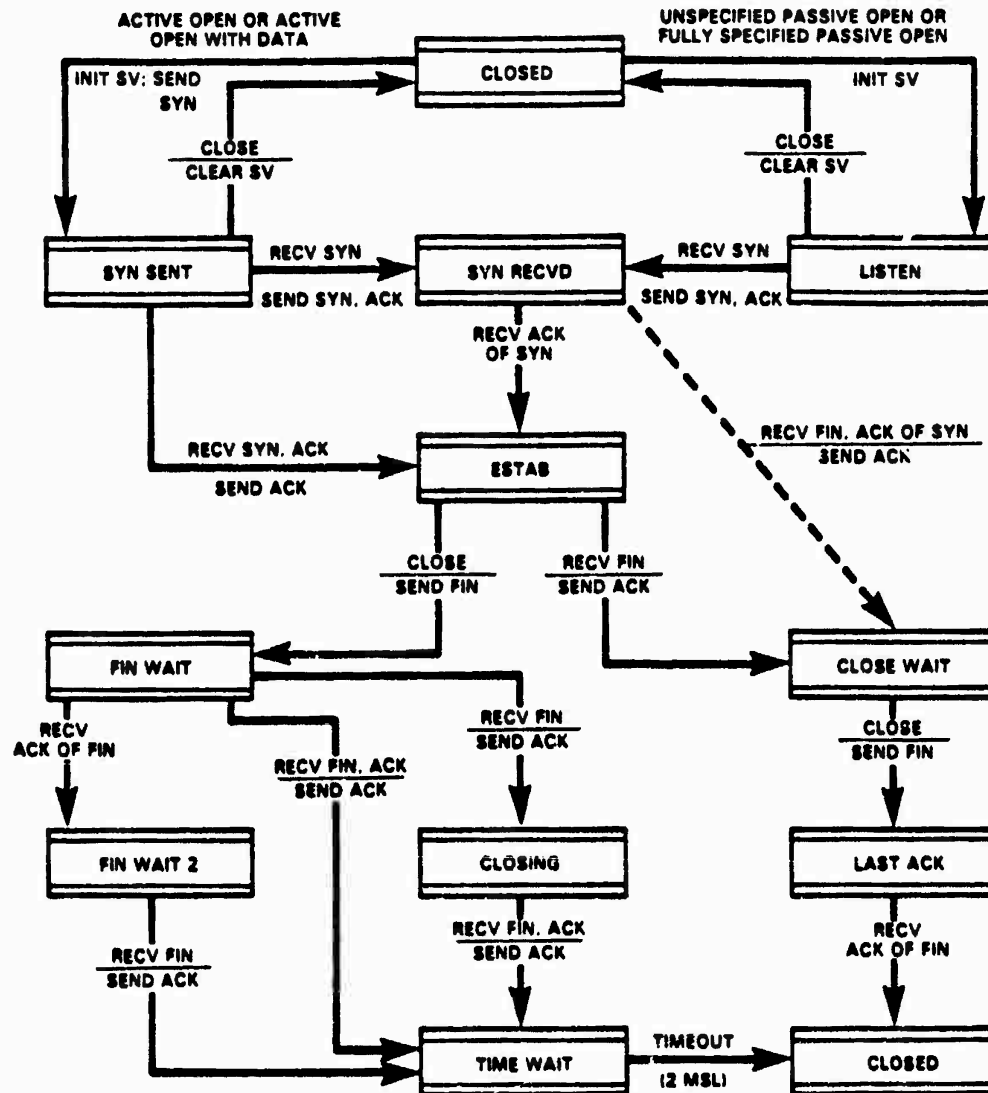


FIGURE 13. TCP entity state summary.

----- LEGEND -----
 recv - NET_DELIVER of segment sv - state vector
 send - NET_SEND of segment init - initialize
 2 MSL - 2 max segment lifetimes clear - nullify

Note that the above figure is intended only as a summary and does not supercede the formal definitions that follow.

MIL-STD-1778
12 August 1983

- h. original precedence (sv.original_prec): one of eight levels of special handling requested by the local ULP in the open request.
- i. actual precedence (sv.actual_prec): one of eight levels of special handling negotiated during connection establishment and verified throughout connection lifetime.
- j. security (sv.sec): information (including security level, compartment, handling restrictions, and transmission control code) defined by the local ULP.
- k. sec_ranges: security structure which specifies the allowed ranges in compartment, handling restrictions, transmission control codes and security levels.
- l. ulp timeout (sv.ulp_timeout): the maximum delay allowed for data transmitted on the connection.
- m. ULP timeout action: in the event of a ULP timeout, determines if the connection is terminated or an error is reported to the ULP.
- n. send unacknowledged sequence number (sv.send_una): oldest unacknowledged send sequence number (i.e. left edge of send window).
- o. send next sequence number (sv.send_next): sequence number of the next data octet to be sent.
- p. send free sequence number (sv.send_free): sequence number of the first free octet in the send queue (i.e. the next octet to be received from the local ULP).
- q. send window (sv.send_wndw): allowed number of octets that may be sent to the remote TCP relative to the send unacknowledged sequence number.
- r. send urgent sequence number (sv.send_urg): sequence number of the last octet of urgent data in send stream.
- s. send push sequence number (sv.send_push): sequence number of the last octet of pushed data in the send stream.
- t. send last window update 1 (sv.send_lastup1): sequence number of the incoming segment used for last window update.
- u. send last window update 2 (sv.send_lastup2): acknowledgment number of the incoming segment used for last window update.
- v. send initial sequence number (sv.send_isn): sequence number of the original SYN sent.

MIL-STD-1778
12 August 1983

- w. send fin flag (sv.send_finflag): indicates that the local ULP has issued a Close request.
- x. send maximum segment size (sv.send_max_seg): maximum sized segment to be sent to the remote TCP on this connection.
- y. send queue (sv.send_queue): location of data received from the local ULP and either awaiting acknowledgment, or awaiting transmission. This area is accessed only by the data management routines.
- z. receive next sequence number (sv.rcv_next): sequence number of next data octet expected to be received.
- aa. receive save sequence number (sv.rcv_next): sequence number of next data octet to be delivered to the local ULP.
- bb. receive window (sv.rcv_wndw): allowed number of data octets to be received from the remote TCP starting with the receive next sequence number.
- cc. receive alloc (sv.rcv_alloc): the number of data octets that will be accepted by the local ULP.
- dd. receive urgent sequence number (sv.rcv_urg): sequence number of the last octet of urgent data in receive stream.
- ee. receive push sequence number (sv.rcv_push): sequence number of the last octet of pushed data in receive stream.
- ff. receive initial sequence number (sv.rcv_isn): sequence number of the SYN received from remote TCP.
- gg. receive fin flag (sv.rcv_finflag): indicates that fin has been received from the remote TCP.
- hh. receive queue (sv.rcv_queue): location of data accepted from remote TCP before delivery to local ULP. This area is accessed only by the data management routines.

9.4.4 Data structures. The TCP entity state machine references certain data areas corresponding to the state vector, the service requests and responses on the upper interface, and the service requests and responses on the lower interface. For clarity in the events and actions section, these data structures are declared in ADA. However, a data structure may be partially typed or untyped where specific formats or data types are implementation dependent.

9.4.4.1 State vector. The TCP entity state vector is defined in paragraph 9.4.1 above. The corresponding structure is declared as:

MIL-STD-1778
12 August 1983

```

sv: state_vector_type;

type state_vector_type is
  record
    state: (CLOSED, LISTEN, SYN_RECVD, SYN_SENT,
            ESTAB, FIN_WAIT1, FIN_WAIT2,
            CLOSE_WAIT, CLOSING, LAST_ACK, TIME_WAIT);
    source_addr: address_type;
    source_port: TWO_OCTETS;
    destination_addr: address_type;
    destination_port: TWO_OCTETS;
    lcn: integer;
    open_mode: (ACTIVE, PASSIVE);
    original_prec: precedence_type;
    actual_prec: precedence_type;
    sec: security_type;
    sec_ranges: security_struct;
    ULP_timeout: integer;
    ULP_timeout_action: integer;
    send_una: sequence_number_type;
    send_next: sequence_number_type;
    send_free: sequence_number_type;
    send_vndw: integer;
    send_urg: sequence_number_type;
    send_push: sequence_number_type;
    send_lastup1: sequence_number_type;
    send_lastup2: sequence_number_type;
    send_isn: sequence_number_type;
    send_finflag: boolean;
    send_max_seg: integer;
    send_queue: timed_queue_type;
    rcv_next: sequence_number_type;
    rcv_save: sequence_number_type;
    rcv_vndw: integer;
    rcv_alloc: integer;
    rcv_urg: sequence_number_type;
    rcv_push: sequence_number_type;
    rcv_isn: sequence_number_type;
    rcv_finflag: boolean;
    rcv_queue: queue_type;
  end record;

```

9.4.4.2 From ULP. The from_ULP structure holds the service request parameters and data associated with the service request primitives as specified in paragraph 6.3. The from_ULP structure is declared as:

```

type from_ULP_type is
  record
    request_name: (Unspecified_Passive_Open, Full_Passive_Open,
                  Active_Open, Active_Open_with_data,
                  Send, Allocate, Close, Abort, Status);

```

MIL-STD-1778
12 August 1983

```
source_addr
source_port
destination_addr
destination_port
lcn
ULP_timeout
ULP_timeout_action
precedence
security
sec_ranges
data
data_length
push_flag
urgent_flag
end record;
```

9.4.4.3 To ULP. The to_ULD structure holds service response parameters and data as specified in paragraph 6.4. Although the structure is composed of the parameters from all the service requests, a particular service response will use only those structure elements corresponding to its specified parameters. The to_ULD structure is declared as:

```
type to_ULD_type is
  record
    service_response : (OPEN_ID, OPEN_FAIL, OPEN_SUCCESS,
                        DELIVER, CLOSING, TERMINATE, ERROR);

    source_addr
    source_port
    destination_addr
    destination_port
    lcn
    data
    data_length
    urgent_flag
    error_desc
    status_block: status_block_type;
  end record;
```

```
type status_block_type is
  record
    connection_state
    send_window
    receive_window
    amount_of_unacked_data
    amount_of_unreceived_data
    urgent_state
    precedence
    security
    sec_ranges
    ULP_timeout
    ULP_timeout_action
  end record;
```

MIL-STD-1778
12 August 1983

9.4.4.4 To NET. The to NET structure holds the service request parameters and data associated with the NET SEND service request specified in paragraph 8.2. This structure directly corresponds to the to NET structure declared in paragraph 8.3.2 of the lower layer service requirements section. The to NET structure is declared as:

```
type to NET_type is  
record  
    source_addr  
    destination_addr  
    protocol  
    type_of_service is  
        record  
            precedence  
            reliability  
            delay  
            throughput  
            reserved  
        end record;  
    time_to_live  
    dont_fragment  
    length  
    seg: segment_type;  
    options  
end record;
```

9.4.4.5 From NET. The from NET structure holds the service response parameters and data associated with the NET DELIVER service response, as specified in paragraph 8.2.2. This structure directly corresponds to the from NET structure declared in paragraph 8.3.3 of the lower layer service requirements section. The from NET structure is declared as:

```
type from NET_type is  
record  
    source_addr  
    destination_addr  
    protocol  
    type_of_service is  
        record  
            precedence  
            reliability  
            delay  
            throughput  
            reserved  
        end record;  
    length  
    seg: segment_type;  
    options  
    error  
end record;
```

MIL-STD-1778
12 August 1983

9.4.4.6 Segment type. A segment_type structure holds a TCP segment made up of a header portion and a data portion as specified in Section 9.3. A segment_type structure is declared as:

```
type segment_type is  
  record  
    source_port      : TWO_OCTETS;  
    destination_port : TWO_OCTETS;  
    seq_num          : FOUR_OCTETS;  
    ack_num          : FOUR_OCTETS;  
    data_offset      : HALF_OCTET;  
    reserved         : SIX_EIGHTHS_OCTET;  
    urg_flag         : ONE_BIT;  
    ack_flag         : ONE_BIT;  
    push_flag        : ONE_BIT;  
    rst_flag         : ONE_BIT;  
    syn_flag         : ONE_BIT;  
    fin_flag         : ONE_BIT;  
    wndw             : TWO_OCTETS;  
    checksum         : TWO_OCTETS;  
    urgptr           : TWO_OCTETS;  
    options          : is array of OCTET;  
    padding          : is array of OCTET;  
    data             : is array of OCTET;  
  end record;
```

9.4.4.7 Supplemental type declarations.

```
type address_type is FOUR_OCTETS;  
type sequence_number_type is FOUR_OCTETS;  
type precedence_type is INTEGER range 0..7;  
type security_type is  
  record  
    security_level : HALF_OCTET;  
    compartment    : TWO_OCTETS;  
    handling       : TWO_OCTETS;  
    trans_control_code : THREE_OCTETS;  
  end record;  
  
subtype OCTET is INTEGER range 0..255;  
subtype HALF_OCTET is INTEGER range 0..15;  
subtype FIVE_EIGHTHS_OCTET is INTEGER range 0..31;  
subtype TWO_OCTETS is INTEGER range 0..2**16-1;  
subtype THREE_OCTETS is INTEGER range 0..2**24-1;  
subtype FOUR_OCTETS is INTEGER range 0..2**32-1;  
NULL_RESERVED      : constant FIVE_EIGHTHS_OCTET := 0;  
OPTIONLESS_HEADER  : constant INTEGER := 5;  
NORMAL             : constant INTEGER := 0;  
NULL               : constant INTEGER := 0;  
  --NULL assumed to be outside the sequence number space.  
DEFAULT_PRECEDENCE : constant INTEGER := 0;
```


MIL-STD-1778
12 August 1983

```

DEFAULT_PRECEDENCE      : constant INTEGER := 0;
DEFAULT_TIMEOUT         : constant INTEGER := 01111000(8);
DEFAULT_TIMEOUT_ACTION  : constant INTEGER := 1;
DEFAULT_SEC_LIST        : security_list;
ONE_MINUTE_TTL          : constant INTEGER := 00111100(8);
THIS_ADDRESS            : constant INTEGER;  --impl. dependant
TCP_ID                  : constant INTEGER;  --reference [5]

```

9.4.5 Event list. The events for the TCP entity state machine are drawn from the service request primitives defined in the service definition of Section 6.2. Optional service request parameters are shown in brackets. The capitalized list of parameters represent the actual values of the parameters passed by the service primitive. The event list:

- a. Unspecified Passive Open (SOURCE_PORT,
[,TIMEOUT] [,TIMEOUT_ACTION]
[,PRECEDENCE] [,SEC_RANGES]);
- b. Full Passive Open (SOURCE_PORT,
DESTINATION_PORT, DESTINATION_ADDRESS,
[,TIMEOUT] [,TIMEOUT_ACTION]
[,PRECEDENCE] [,SEC_RANGES]);
- c. Active Open (SOURCE_PORT,
DESTINATION_PORT, DESTINATION_ADDRESS
[,TIMEOUT] [,TIMEOUT_ACTION]
[,PRECEDENCE] [,SECURITY]);
- d. Active Open w/data (SOURCE_PORT,
DESTINATION_PORT, DESTINATION_ADDRESS
[,TIMEOUT] [,TIMEOUT_ACTION] [,PRECEDENCE]
[,SECURITY]); DATA, DATA_LENGTH, PUSH_
FLAG, URGENT_FLAG);
- e. Send (LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG [,TIMEOUT]);
- f. Allocate (LCN, DATA_LENGTH)
- g. Close (LCN)
- h. Abort (LCN)
- i. Status (LCN)
- j. NET_DELIVER (SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)
- k. Retransmission Timeout
- l. ULP Timeout
- m. Time Wait Timeout

MIL-STD-1778
12 August 1983

9.4.6 Events and actions. This section is organized in three parts. The first part contains a decision table representation of state machine events and actions. The decision tables are organized by state; each table corresponds to one event. The second part specifies the decision functions appearing at the top of each column of a decision table. These functions examine attributes of the event and the state vector to return a set of decision results. The results become the elements of each column. The third part specifies action procedures appearing at the right of every row. Each row of the decision table combines the decision results to determine appropriate event processing. These procedures specify event processing algorithms in detail.

9.4.6.1 Decision tables. The Status event can occur in any state except closed; TCP's action is to return the current state vector information as specified in the STATUS RESPONSE service response. If the primary state vector element is not changed in the decision table row corresponding to an event, the "primary" state remains unchanged. The checksum is assumed to be computed for all incoming segments. When the computed checksum does not match the segment's header checksum field, the segment is discarded without being acknowledged.

9.4.6.1.1 State = closed.

Legend

d = "don't care" condition

Event: Active Open (LOCAL_PORT, REMOTE_PORT, REMOTE_ADDRESS
[TIMEOUT] [TIMEOUT_ACTION] [PRECEDENCE] [SECURITY])

TABLE I. Active open event in a closed state.

Actions:

RESOURCES SUFFIC OPEN?	SEC PREC ALLOWED	
NO	d	ERROR ("INSUFFICIENT RESOURCES")
YES	NO	ERROR ("SECURITY/PRECEDENCE NOT ALLOWED")
YES	YES	OPEN. GEN_SYNIALONEI. SV STATE = SYN_SENT

Event: Active Open with Data (LOCAL_PORT, REMOTE_PORT, REMOTE_ADDRESS,
[TIMEOUT] [TIMEOUT_ACTION] [PRECEDENCE]
[SECURITY] DATA, DATA_LENGTH, PUSH_FLAG,
URGENT_FLAG)

MIL-STD-1778
12 August 1983

TABLE II. Active open with data event in a closed state.

Actions:

RESOURCES SUFFIC OPEN?	SEC PREC ALLOWED	
NO	d	ERROR ("INSUFFICIENT RESOURCES")
YES	NO	ERROR ("SECURITY/PRECEDENCE NOT ALLOWED.")
YES	YES	OPEN. GEN_SYN (WITH_DATA); SV. STATE = SYN_SENT

Event: Full Passive Open (LOCAL PORT, REMOTE PORT, REMOTE ADDRESS,
[TIMEOUT] [TIMEOUT_ACTION] [PRECEDENCE] [SEC_RANGES])

TABLE III. Full passive open event in a closed state.

Actions:

RESOURCES SUFFIC OPEN?	SEC PREC ALLOWED	
NO	d	ERROR ("INSUFFICIENT RESOURCES")
YES	NO	ERROR ("SECURITY/PRECEDENCE NOT ALLOWED")
YES	YES	OPEN. SV STATE = LISTEN

Event: Unspecified Passive Open (LOCAL PORT, [TIMEOUT] [TIMEOUT_ACTION]
[PRECEDENCE] [SEC_RANGES])

TABLE IV. Unspecified passive open event in a closed state.

Actions:

RESOURCES SUFFIC OPEN?	SEC PREC ALLOWED	
NO	d	ERROR ("INSUFFICIENT RESOURCES")
YES	NO	ERROR ("SECURITY/PRECEDENCE NOT ALLOWED")
YES	YES	OPEN. SV STATE = LISTEN

Event: Send ()
or Close ()
or Abort ()
or Allocate ()

Actions: error ("Connection does not exist.")

MIL-STD-1778
12 August 1983

Event: NET_DELIVER (SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

TABLE V. Net deliver event in a closed state.

Actions:

=====

RST ON ?	ACK ON ?	
NO	NO	RESET (SEQ)
NO	YES	RESET (SEQ)
YES	d	-- NO ACTION

9.4.6.1.2 State = listen.

Event: Close (LCN)
or Abort (LCN)

Actions: reset_self(UC); sv.state=CLOSED

Event: Allocate (LCN, DATA_LENGTH)

Actions: new_allocation

Event: Send ()

Actions: error ("Illegal request.")

Event: Active Open ()
or Active Open with data ()
or Full Passive Open ()
or Unspecified Passive Open ()

Actions: error ("Connection already exists.")

Event: NET_DELIVER (SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

MIL-STD-1778
12 August 1983

TABLE VI. Net deliver event in a listen state.

Actions:

=====

RST ON ?	ACK ON ?	SYN ON ?	SEC MATCH ?	SV PREC VS SEG PREC	
NO	NO	NO	d	d	-- NO ACTION
NO	NO	YES	NO	d	RESET (SEG)
NO	NO	YES	YES	GREATER OR EQUAL	RECORD_SYN: GEN_SYN (WITH_ACK); SV. STATE = SYN_RECVD
NO	NO	YES	YES	LESS	RECORD_SYN: RAISE_PREC: GEN_SYN (WITH_ACK); SV. STATE = SYN_RECVD
NO	YES	d	d	d	RESET (SEG)
YES	d	d	d	d	-- NO ACTION

9.4.6.1.3 State = SYN SENT.

Event: Close (LCN)
or Abort (LCN)

Actions: reset_self(UC); sv.state=CLOSED

Event: Send (LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG [TIMEOUT])
[TIMEOUT_ACTION]

TABLE VII. Close or abort event in a SYN SENT state.

Actions:

=====

RESOURCES SUFFIC BENO?	
NO	ERROR: "INSUFFICIENT RESOURCES"
YES	SAVE_BENO_DATA

Event: Allocate (LCN, DATA_LENGTH)

Actions: new_allocation

Event: Active Open ()
or Active Open with data ()
or Full Passive Open ()
or Unspecified Passive Open ()

Actions: error ("Connection already exists.")
106

MIL-STD-1778
12 August 1983

Event: Retransmission Timeout

Actions: retransmit

Event: ULP Timeout

Actions: if (ULP_timeout_action = 1)
then open_fail; sv.state = CLOSED;
else report_timeout;

Event: NET_DELIVER (SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

TABLE VIII. Net deliver event in a SYN SENT state.

Actions:

ACK STATUS TEST 1	RST ON ?	SEC MATCH ?	SV PREC VS SEG PREC	SYN ON ?	FIN ON ?	
NONE	NO	NO	d	d	d	RESET (SEQ)
NONE	NO	YES	d	NO	d	-- NO ACTION
NONE	NO	YES	GREATER OR EQUAL	YES	NO	RECORD_SYN; SEND_ACK (SV.RECV_ISN + 1); SV.STATE = SYN_RECVD
NONE	NO	YES	GREATER OR EQUAL	YES	YES	RECORD_SYN; SEND_ACK (SV.RECV_ISN + 1); SAVE_FIN; SV.STATE = SYN_RECVD
NONE	NO	YES	LESS	YES	NO	RECORD_SYN; RAISE_PREC; SEND_ACK (SV.RECV_ISN + 1); SV.STATE = SYN_RECVD
NONE	NO	YES	LESS	YES	YES	RECORD_SYN; RAISE_PREC; SEND_ACK (SV.RECV_ISN + 1); SAVE_FIN; SV.STATE = SYN_RECVD
NONE	YES	d	d	d	d	-- NO ACTION
INVAL	NO	d	d	d	d	RESET (SEQ)
INVAL	YES	d	d	d	d	-- NO ACTION
VALID	NO	NO	d	d	d	RESET (SEQ)
VALID	NO	YES	GREATER	d	d	RESET (SEQ)
VALID	NO	YES	LESS OR EQUAL	NO	d	-- NO ACTION
VALID	NO	YES	LESS OR EQUAL	YES	NO	RAISE_PREC; CONN_OPEN; SV.STATE = ESTAB
VALID	NO	YES	LESS OR EQUAL	YES	YES	RAISE_PREC; CONN_OPEN; SET_FIN; SV.STATE = CLOSE_WAIT
VALID	YES	d	d	d	d	OPENFAIL; SV.STATE = CLOSED

MIL-STD-1778
12 August 1983

9.4.6.1.4 State = SYN RECVD.

Event: Close (LCN)

Actions: send_fin; sv.state=FIN_WAIT1

Event: Abort (LCN)

Actions: reset (CURRENT); reset_self(UA); sv.state=CLOSED

Event: Send (LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG [TIMEOUT]
[TIMEOUT_ACTION])

TABLE IX. Send event in a SYN RECVD state.

Actions:

=====

RESOURCES SUFFIC SEND?	
NO	ERROR ("INSUFFICIENT RESOURCES.")
YES	SAVE_SEND_DATA

Event: Allocate (LCN, DATA_LENGTH)

Actions: new_allocation

Event: Active Open ()
or Active Open with data ()
or Full Passive Open ()
or Unspecified Passive Open ()

Actions: error ("Connection already exists.")

Event: Retransmission Timeout

Actions: retransmit

Event: ULP Timeout

Actions: if (ULP_timeout_action = 1)
then reset (CURRENT); openfail; sv.state=CLOSED
else report_timeout;

MIL-STD-1778
12 August 1983

Event: NET_DELIVER (SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

TABLE X. Net deliver event in a SYN RECVD state.

Actions:

SEQ# STATUS	RST ON?	SEC PREC MATCH?	OPEN MODE?	SYN IN WINDOW	ACK STATUS TEST 1	ZERO RCV WINDOW	FIN SEEN?	
INVAL	NO	d	d	d	d	d	d	SEND_ACK (SV_RECV_NEXT)
INVAL	YES	d	d	d	d	d	d	-- NO ACTION
VALID	NO	NO	PASS	d	d	d	d	RESET (SEG) PART_RESET: SV.STATE = LISTEN
VALID	NO	NO	ACT	d	d	d	d	RESET (SEG); OPENFAIL: SV.STATE = CLOSED
VALID	NO	YES	d	NO	NONE	d	d	-- NO ACTION
VALID	NO	YES	d	NO	INVAL	d	d	RESET (SEG)
VALID	NO	YES	d	NO	VALID	NO	NO	CONN_OPEN: SV.STATE = ESTAB
VALID	NO	YES	d	NO	VALID	NO	YES	CONN_OPEN: SET_FIN: SV.STATE = CLOSE_WAIT
VALID	NO	YES	d	NO	VALID	YES	d	UPDATE: CHECK_URG: SV.STATE = ESTAB
VALID	NO	YES	d	YES	d	d	d	RESET (SEG); OPENFAIL: SV.STATE = CLOSED
VALID	YES	d	PASS	d	d	d	d	PART_RESET: SV.STATE = LISTEN
VALID	YES	d	ACT	d	d	d	d	OPENFAIL: SV.STATE = CLOSED

9.4.6.1.5 State = ESTAB.

Event: Close (LCN)

Actions: send_fin; sv.state=FIN_WAIT!

Event: Abort (LCN)

Actions: reset(CURRENT); reset_self(UA); sv.state=CLOSED

MIL-STD-1778
12 August 1983

Event: Send (LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG [TIMEOUT]
[TIMEOUT_ACTION])

TABLE XI. Send event in an estab state.

Actions:

RESOURCES SUFFIC SEND?	
NO	ERROR ("INSUFFICIENT RESOURCES")
YES	DISPATCH

Event: Allocate (LCN, DATA_LENGTH)

Actions: new_allocation

Event: Active Open ()
or Active Open with data ()
or Full Passive Open ()
or Unspecified Passive Open ()

Actions: error ("Connection already exists.")

Event: Retransmission Timeout

Actions: retransmit

Event: ULP Timeout

Actions: if (ULP_timeout_action = 1)
then reset (CURRENT); reset_self(UT); sv.state=CLOSED
else report_timeout

Event: NET_DELIVER (SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

MIL-STD-1778
12 August 1983

TABLE XII. Net deliver event in an estab state.

Actions:

SEQ# STATUS ?	RST ON ?	SEC PREC MATCH?	SYN IN WINDOW	ACK STATUS TEST 2?	
INVAL	NO	d	d	d	SEND_ACK (SV, RECV_NEXT)
INVAL	YES	d	d	d	-- NO ACTION
VALID	NO	NO	d	d	RESET (SEG); RESET_SELF (SP); SV STATE = CLOSED
VALID	NO	YES	NO	NONE	-- NO ACTION
VALID	NO	YES	NO	INVAL	SEND_ACK (SV, RECV_NEXT)
VALID	NO	YES	NO	VALID	UPDATE
VALID	NO	YES	YES	d	RESET (SEG); RESET_SELF (SP); SV STATE = CLOSED
VALID	YES	d	d	d	RESET_SELF (RA); SV STATE = CLOSED

9.4.6.1.6 State = CLOSE WAIT.

Event: Send (LCN, DATA, DATA_LENGTH, PUSH_FLAG, URGENT_FLAG [TIMEOUT]
[TIMEOUT_ACTION])

TABLE XIII. Send event in a CLOSE WAIT state.

Actions:

RESOURCES SUFFIC SEND?	
NO	ERROR ("INSUFFICIENT RESOURCES")
YES	DISPATCH

MIL-STD-1778
12 August 1983

Event: Active Open ()
or Active Open with data ()
or Full Passive Open ()
or Unspecified Passive Open ()

Actions: error ("Connection already exists.")

Event: Close (LCN)

Actions: send_fin; sv.state=LAST_ACK

Event: Abort (LCN)

Actions: reset(CURRENT); reset_self(UA); sv.state=CLOSED

Event: Retransmission Timeout

Actions: retransmit

Event: ULP Timeout

Actions: if (ULP_timeout_action = 1)
then reset(CURRENT); reset_self(UT); sv.state=CLOSED
else report_timeout

Event: NET_DELIVER (SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

TABLE XIV. Net deliver event in a CLOSE WAIT state.

Actions:

SEQ STATUS '	RST ON '	SEC PREC MATCH'	SYN IN WINDOW	ACK STATUS TEST 2'	ZERO RCV WINDOW	FIN SEEN '	
INVAL	NO	d	d	d	d	d	SEND_ACK (SV_RECV_NEXT)
INVAL	YES	d	d	d	d	d	-- NO ACTION
VALID	NO	NO	d	d	d	d	RESET (SEG); RESET_SELF (SPI); SV STATE = CLOSED
VALID	NO	YES	NO	NONE	d	d	-- NO ACTION
VALID	NO	YES	NO	INVAL	d	d	SEND_ACK (SV_RECV_NEXT)
VALID	NO	YES	NO	VALID	NO	NO	UPDATE ACCEPT
VALID	NO	YES	NO	VALID	NO	YES	UPDATE ACCEPT; SET_FIN SV STATE = CLOSE_WAIT
VALID	NO	YES	NO	VALID	YES	d	UPDATE CHECK_URG
VALID	NO	YES	YES	d	d	d	RESET (SEG); RESET_SELF (SPI); SV STATE = CLOSED
VALID	YES	d	d	d	d	d	RESET_SELF (RA); SV STATE = CLOSED

MIL-STD-1778
12 August 1983

9.4.6.1.7 State = closing.

Event: Allocate (LCN, DATA_LENGTH)

Actions: new_allocation

Event: Send ()
or Close ()

Actions: error ("Connection closing.")

Event: Active Open ()
or Active Open with data ()
or Full Passive Open ()
or Unspecified Passive Open ()

Actions: error (Connection already exists.)

Event: Abort (LCN)

Actions: reset (CURRENT); reset_self(UA); sv.state=CLOSED;

Event: Retransmission Timeout

Actions: retransmit

Event: ULP Timeout

Actions: if (ULP timeout-action = 1)
reset (CURRENT); reset_self(UT); sv.state=CLOSED
else report_timeout

Event: NET_DELIVER(SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

MIL-STD-1778
12 August 1983

TABLE XV. Net deliver event in a closing state.

Actions:

SEQ# STATUS ?	RST ON ?	SEC PREC MATCH?	SYN IN WINDOW	ACK STATUS TEST 2?	FIN ACK'D ?	
INVAL	NO	d	d	d	d	SEND_ACK (SV, RECV_NEXT)
INVAL	YES	d	d	d	d	-- NO ACTION
VALID	NO	NO	d	d	d	RESET (SEQ); RESET_SELF (SP); SV, STATE = CLOSED
VALID	NO	YES	NO	NONE	d	-- NO ACTION
VALID	NO	YES	NO	INVAL	d	SEND_ACK (SV, RECV_NEXT)
VALID	NO	YES	NO	VALID	NO	UPDATE
VALID	NO	YES	NO	VALID	YES	START_TIME_WAIT; SV STATE = TIME_WAIT
VALID	NO	YES	YES	d	d	RESET (SEQ); RESET_SELF (SP); SV, STATE = CLOSED
VALID	YES	d	d	d	d	RESET_SELF (RA); SV, STATE = CLOSED

9.4.6.1.8 State = FIN WAIT1.

Event: Allocate(LCN, DATA_LENGTH)

Actions: new_allocation

Event: Send()
or Close()

Actions: error("Connection closing.")

Event: Active Open()
or Active Open with data()
or Full Passive Open()
or Unspecified Passive Open()

Actions: error("Connection already exists.")

MIL-STD-1778
12 August 1983

Event: Abort(LCN)

Actions: reset(CURRENT); reset_self(UA); sv.state=CLOSED

Event: Retransmission Timeout

Actions: retransmit

Event: ULP Timeout

Actions: if (ULP_timeout action = 1)
then reset(CURRENT); reset_self(UT); sv.state=CLOSED
else report_timeout

Event: NET_DELIVER(SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

TABLE XVI. NET deliver event in a FIN WAIT1 state.

Actions:

SEQ# STATUS ?	RST ON ?	SEC PREC MATCH?	SYN IN WINDOW	ACK STATUS TEST 2?	ZERO RCV WINDOW	FIN ACK'D ?	FIN ON ?	
INVAL	NO	d	d	d	d	d	d	SEND_ACK (SV RCV_NEXT)
INVAL	YES	d	d	d	d	d	d	-- NO ACTION
VALID	NO	NO	d	d	d	d	d	RESET, RESET_SELF (SPI), SV STATE = CLOSED
VALID	NO	YES	NO	NONE	d	d	d	-- NO ACTION
VALID	NO	YES	NO	INVAL	d	d	d	SEND_ACK (SV RCV_NEXT)
VALID	NO	YES	NO	VALID	NO	NO	NO	UPDATE, ACCEPT
VALID	NO	YES	NO	VALID	NO	NO	YES	UPDATE, ACCEPT, SET_FIN, SV STATE = CLOSING
VALID	NO	YES	NO	VALID	NO	YES	NO	UPDATE, ACCEPT, SV STATE = FIN_WAIT 2
VALID	NO	YES	NO	VALID	NO	YES	YES	UPDATE, ACCEPT, SET_FIN, START_TIME_WAIT SV STATE = TIME_WAIT
VALID	NO	YES	NO	VALID	Y	NO	d	UPDATE
VALID	NO	YES	NO	VALID	YES	YES	d	UPDATE, SV STATE = FIN_WAIT 2
VALID	NO	YES	YES	d	d	d	d	RESET (SEQ), RESET_SELF (SPI), SV STATE = CLOSED
VALID	YES	d	d	d	d	d	d	RESET_SELF (RA), SV STATE = CLOSED

MIL-STD-1778
12 August 1983

9.4.6.1.9 State = FIN WAIT2.

Event: Abort(LCN)

Actions: reset(CURRENT); reset_self(UA); sv.state=CLOSED

Event: Allocate(LCN, DATA_LENGTH)

Actions: new_allocation

Event: Send()
or Close()

Actions: error("Connection closing.")

Event: Active Open()
or Active Open with data()
or Full Passive Open()
or Unspecified Passive Open()

Actions: error("Connection already exists.")

Event: Retransmission Timeout

Actions: retransmit

Event: ULP Timeout

Actions: if (ULP_timeout_action = 1)
then reset(CURRENT); reset_self(UT); sv.state=CLOSED
else report_timeout

Event: NET_DELIVER(SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

MIL-STD-1778
12 August 1983

TABLE XVII. Net deliver event in a FIN WAIT2 state.

Actions:

SEQ# STATUS ?	RST ON ?	SEC PREC MATCH?	SYN IN WINDOW	ACK STATUS TEST 2?	ZERO RCV WINDOW	FIN ON ?	
INVAL	NO	d	d	d	d	d	SEND_ACK (SV, RCV_NEXT)
INVAL	YES	d	d	d	d	d	-- NO ACTION
VALID	NO	NO	d	d	d	d	RESET (SEG); RESET_SELF (SP); SV, STATE = CLOSED
VALID	NO	YES	NO	NONE	d	d	-- NO ACTION
VALID	NO	YES	NO	INVAL	d	d	SEND_ACK (SV, RCV_NEXT)
VALID	NO	YES	NO	VALID	NO	NO	UPDATE: ACCEPT
VALID	NO	YES	NO	VALID	NO	YES	UPDATE: ACCEPT; SET_FIN; START_TIME_WAIT; SV, STATE = TIME_WAIT
VALID	NO	YES	NO	VALID	YES	d	UPDATE
VALID	NO	YES	YES	d	d	d	RESET (SEG); RESET_SELF (SP); SV, STATE = CLOSED
VALID	YES	YES	d	d	d	d	RESET_SELF (RA); SV, STATE = CLOSED

9.4.6.1.10 State = last ACK.

Event: Abort(LCN)

Actions: reset_self(UA); sv.state=CLOSED

Event: Send()
or Close()
or Allocate()

Actions: error("Connection closing.")

Event: Active Open()
or Active Open with data()
or Full Passive Open()
or Unspecified Passive Open()

Actions: error("Connection already exists.")

MIL-STD-1778
12 August 1983

Event: Retransmission Timeout

Actions: retransmit

Event: ULP Timeout

Actions: if (ULP_timeout action = 1)
then reset(CURRENT); reset_self(UT); sv.state=CLOSED
else report_timeout

Event: NET_DELIVER(SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

TABLE XVIII. Net deliver event in a LAST ACK state.

Actions:

=====

SEQ/STATUS ?	RST ON ?	SEC PREC MATCH?	SYN IN WINDOW	ACK STATUS TEST 2?	FIN ACK'D ?	
INVAL	NO	d	d	d	d	SEND_ACK (SV_RECV_NEXT)
INVAL	YES	d	d	d	d	-- NO ACTION
VALID	NO	NO	d	d	d	RESET (SEQ); RESET_SELF (SPI); SV.STATE = CLOSED
VALID	NO	YES	NO	NONE	d	-- NO ACTION
VALID	NO	YES	NO	INVAL	d	SEND_ACK (SV_RECV_NEXT)
VALID	NO	YES	NO	VALID	NO	-- NO ACTION
VALID	NO	YES	NO	VALID	YES	RESET_SELF (UCI); SV.STATE = CLOSED
VALID	YES	YES	YES	d	d	RESET (SEQ); RESET_SELF (SPI); SV.STATE = CLOSED
VALID	YES	d	d	d	d	RESET_SELF (RAI); SV.STATE = CLOSED

9.4.6.1.11 State = TIME WAIT.

Event: Abort(LCN)

Actions: reset_self(UA); sv.state=CLOSED

MIL-STD-1778
12 August 1983

Event: Send()
or Close()
or Allocate()

Actions: error("Connection closing.")

Event: Active Open()
or Active Open with data()
or Full Passive Open()

or Unspecified Passive Open()

Actions: error("Connection already exists.")

Event: Time Wait Timeout

Actions: reset_self(UC); sv.state=CLOSED

Legend

d = "don't care" condition

Event: NET_DELIVER(SOURCE_ADDRESS, DESTINATION_ADDRESS, PROTOCOL,
TOS[precedence, reliability, delay, throughput],
OPTIONS[security], LENGTH, DATA)

TABLE XIX. Net deliver event in a TIME WAIT state.

Actions:

SEQ# STATUS ?	RST ON ?	SEC PREC MATCH?	SYN IN WINDOW	ACK STATUS TEST ?	FIN ON ?	
INVAL	NO	d	d	d	d	SEND_ACK (SV RECV_NEXT)
INVAL	YES	d	d	d	d	-- NO ACTION
VALID	NO	NO	d	d	d	RESET (SEQ); RESET_SELF (SP); SV STATE = CLOSED
VALID	NO	YES	NO	NONE	d	-- NO ACTION
VALID	NO	YES	NO	INVAL	d	SEND_ACK (SV RECV_NEXT)
VALID	NO	YES	NO	VALID	NO	-- NO ACTION
VALID	NO	YES	NO	VALID	YES	SEND_ACK (SV RECV_NEXT); RESTART_TIME_WAIT
VALID	NO	YES	YES	d	d	RESET (SEQ); RESET_SELF (SP); SV STATE = CLOSED
VALID	YES	d	d	d	d	RESET_SELF (RA); SV STATE = CLOSED

MIL-STD-1778
12 August 1983

9.4.6.2 Decision functions. The following functions examine information contained in interface parameters, interface data, and the state vector to make decisions. These decisions can be thought of as further refinements of the event and/or state. The return values of the functions represent decisions made.

9.4.6.2.1 ACK on? The ACK_on function determines whether the acknowledgment field of the incoming segment is in use. The data effects of this function are:

- a. Data examined only: from_NET.seg.ack_flag
 - b. Return values:
 - NO — indicates the ACK flag is false and the ACK number should not be examined
 - YES — indicates that the ACK flag is true and the ACK number is in use
- ```

if (from_NET.seg.ack_flag = TRUE)
 then return (YES)
else return (NO);

```

9.4.6.2.2 ACK status test? The ACK\_status\_test? function compares the ACK number of the incoming segment with the current send variables to determine whether the ACK is valid. This function is intended for use during connection establishment when "old duplicate" ACKs cannot occur. The data effects of this function are:

- a. Data examined only:
  - from\_NET.seg.ack\_num      sv.send\_next
  - from\_NET.seg.ack\_flag    sv.send\_una
- b. Return values:
  - NONE — no ACK appears in the incoming segment
  - INVALID — the incoming segment carries an ACK which is outside the send window
  - VALID — the incoming segment carries an ACK inside the send window which should be used for update

—During connection establishment, an ACK is valid if  
 —it falls inside the send window because old ACKs do not  
 —exist for this connection incarnation.

—Check for presence of ack flag.

```

if (from_NET.seg.ack_flag = FALSE)
 then return (NONE)
else —Validate ACK against current send window

```

```

if (from_NET.seg.ack_num < sv.send_una)
 or (from_NET.seg.ack_num > sv.send_next)

```

---

MIL-STD-1778  
12 August 1983

then return (INVALID ) --present but unacceptable  
else return (VALID); --present and inside send window

9.4.6.2.3 ACK status test2? The ACK\_status\_test2 function examines the ACK number of the incoming segment against the current send variables to determine whether the ACK is valid. This function is intended for use after connection establishment when old duplicate ACKs can legally occur. The data effects of this function are:

- a. Data examined only:  
from\_NET.seg.ack\_flag            sv.send\_next  
from\_NET.seg.ack\_num
- b. Return values:  
NONE     -- no ACK appears in the incoming segment  
INVALID -- the incoming segment carries an ACK for something which has not yet been sent.  
VALID    -- the incoming segment carries an ACK which either falls in the window (and should be used for update) or duplicates a previous ACK.

--After a connection is established, an ACK is valid if  
--it ACKs something sent on this connection incarnation.

--Check for presence of ack flag.

if (from\_NET.seg.ack\_flag = FALSE)  
then return (NONE)  
else --Validate ACK against current send window  
if (from\_NET.seg.ack\_num > sv.send\_next)  
then return (INVALID) --present but unacceptable  
else return (VALID); --present and okay

9.4.6.2.4 Checksum check? The checksum\_check function computes the checksum of an incoming segment and compares it against the checksum field in the header of the incoming segment. The data effects of this function are:

- a. Data examined only:  
all fields of from\_NET.seg            from\_NET.protocol  
from\_NET.source\_addr            from\_NET.length  
from\_NET.destination\_addr

MIL-STD-1778  
12 August 1983

b. Return values:

NO -- indicates that the computed checksum does not match the value in from\_NET.seg.checksum

YES -- indicates that the computed checksum matches the value in from\_NET.seg.checksum

--The checksum algorithm is the 16 bit one's complement of the one's complement sum of all 16 bit words in the segment header and segment text. If a segment contains an odd number of octets, the last octet is padded on the right with zeros to form a 16-bit word for checksum purposes.  
--While computing the checksum, the checksum field itself is replaced with zeros.  
--The checksum includes a 96-bit pseudo header prefixed to the actual TCP header. The pseudo header contains the source address, the destination address, the protocol identifier and the length of the TCP segment (not counting the pseudo header) as passed by the NET\_DELIVER service primitive.

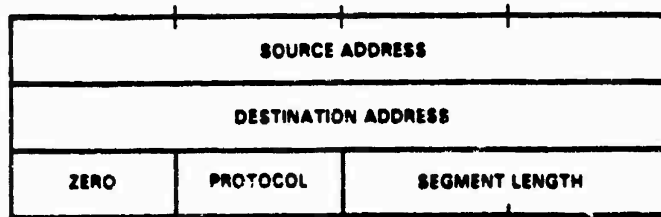


FIGURE 14. Checksum check function.

--The actual computation is implementation dependent.

9.4.6.2.5 FIN ACK'd? The FIN\_ACK'd function examines the acknowledgment field of the incoming segment to determine whether this segment ACKs a previously sent FIN. The data effects of this function are:

a. Data examined only:

|                       |                 |
|-----------------------|-----------------|
| from_NET.seg.ack_flag | sv.send_finflag |
| from_NET.seg.ack_num  | sv.send_next    |

b. Return values:

NO -- the incoming segment does not ACK the FIN

YES -- the incoming segment does carry an ACK of the previously sent FIN

--The sv.send\_finflag indicates that the ULP has issued a CLOSE. The FIN's sequence number is one less than sv.send\_next.

```
if (sv.send_finflag = TRUE)
then
```

---

MIL-STD-1778  
12 August 1983

```
if ((from_NET.seg.ack_flag = TRUE) and
 (from_NET.seg.ack_num = sv.send_next))
then return (YES)
else return (NO)
```

9.4.6.2.6 FIN on? The FIN\_on function determines whether the incoming segment carries a FIN indicating the remote side has no more data to send. The data effects of this function are:

a. Data examined only: from\_NET.seg.fin\_flag

b. Return values:  
NO -- the segment does not contain a FIN  
YES -- the segment does carry a FIN

--The segment header field seg.fin\_flag indicates the  
--presence or absence of a FIN.

```
if (from_NET.seg.fin_flag = FALSE)
then return (NO)
else return (YES);
```

9.4.6.2.7 FIN seen? The FIN\_seen function examines both the incoming segment and the sv.recv variables for the previous or current presence of a FIN from the remote TCP. This function is used in the established state because a FIN may have been recorded during connection opening. The data effects of this function are:

a. Data examined only:  
from\_NET.seg.fin\_flag                      sv.recv\_finflag

b. Return values:  
NO -- No FIN has been received from the remote TCP in this  
or previous segments.

YES -- A FIN has been received, either in the incoming segment  
or a previous segment.

--A FIN received during connection opening is saved in  
--sv.recv\_finflag. A FIN is present in an  
--incoming segment if from\_NET.seg.fin\_flag is set true.

```
if ((sv.recv_finflag = TRUE) or
 (from_NET.seg.fin_flag = TRUE))
then return (YES)
else return (NO);
```

MIL-STD-1778  
12 August 1983

9.4.6.2.8 Open mode? The open\_mode function determines what kind of open service request the local ULP issued. The data effects of this function are:

- a. Data examined only: sv.open\_mode
- b. Return values:
  - ACTIVE -- the ULP requested an Active Open with or without data for this connection.
  - PASSIVE -- the ULP request a Full Passive Open or an Unspecified Passive Open for this connection.

--The type of open request is recorded in sv.open\_mode.

```
if (sv.open_mode = PASSIVE)
then return (PASSIVE)
else return (ACTIVE);
```

9.4.6.2.9 Sv prec vs seg prec? The sv\_prec\_vs\_seg\_prec function compares the precedence recorded in the state vector against the precedence level of the incoming segment. The data effects of this function are:

- a. Data examined only:
  - sv.original\_prec                      from\_NET.type\_of\_service.precedence
- b. Return values:
  - LESS - precedence in sv < segment precedence
  - EQUAL - precedence in sv = segment precedence
  - GREATER - precedence in sv > segment precedence

```
if (sv.original_prec < from_NET.precedence)
then return (LESS)
else if (sv.original_prec = from_NET.precedence)
then return (EQUAL)
else return (GREATER);
```

9.4.6.2.10 Resources suffic open? The resources\_suffic\_open function examines the internal resources available in this TCP entity to determine whether another connection can be supported. The data effects of this function are:

- a. Data examined only: --implementation dependent
- b. Return values:
  - NO -- indicates that another connection cannot be supported at this time.
  - YES -- indicates that internal resources are sufficient to support another connection

MIL-STD-1778  
12 August 1983

--This function is based on the assumption that a TCP entity  
--has finite resources made up of table space, storage capacity,  
--and other implementation dependent areas. Although the amount  
--of these resources may either be fixed at system configuration  
--or vary dynamically according to system usage, more connections  
--may be requested than can be supported by the entity.

if (enough resources are available for another connection)  
then return (YES)  
else return (NO);

9.4.6.2.11 Resources suffice send? The resources\_suffic\_send function examines the resources of this TCP connection to determine if more data can be accepted from the ULP for transfer. The data effects of this function are:

- a. Data examined only: --implementation dependent
- b. Return values:
  - NO -- indicates that data cannot be accepted from the ULP at this time.
  - YES -- indicates that internal resources are sufficient to accept and transfer more ULP data

--This function is based on the assumption that a TCP  
--connection has finite resources. Although the amount of  
--these resources may be fixed at connection establishment,  
--or vary over connection lifetime, at some point a sending  
--ULP might exceed the TCP entity's capacity.

if ( enough resources are available to handle this SEND )  
then return (YES)  
else return (NO);

9.4.6.2.12 RST on? The RST\_on function examines the reset flag of the incoming segment's header to determine the presence of a RST. The data effects of this function are:

- a. Data examined only: from\_NET\_seg\_rst\_flag
- b. Return values:
  - NO -- the incoming segment does not contain a RESET
  - YES -- the incoming segment does carry a RESET

if (from\_NET\_seg\_rst\_flag = FALSE)  
then return (NO)  
else return (YES);

9.4.6.2.13 Sec match? The sec\_match function compares the security parameters (including security level, compartment, transmission control code,



MIL-STD-1778  
12 August 1983

and handling restrictions) defined in the state vector against those accompanying the incoming segment. The data effects of this function are:

- a. Data examined only:  
from\_NET.options[security]      sv.sec
- b. Return values:  
NO -- The values in the state vector do not match those of the incoming segment.  
  
YES -- The security information exactly matches that in the state vector.

--The security information is not carried in the segment header  
--but is passed by the network protocol entity in the  
--NET\_DELIVER option parameter.

```
if (from_NET.options[security] = sv.sec)
then return (YES)
else return (NO);
```

9.4.6.2.14 Sec prec allowed? The sec\_prec\_allowed function examines the security and precedence information requested by a ULP in a connection open request and based on the implementation environment (i.e., secure host, unclassified system, etc.) determines whether this TCP entity can support them. The data effects of this function are:

- a. Data examined only:  
from\_ULP.precedence      from\_ULP.security
  - b. Return values:  
NO -- This TCP entity cannot support the requested security and precedence.  
  
YES -- The security and precedence requested can be supported.
- This decision is implementation dependent.

9.4.6.2.15 Sec range match? The sec\_range\_match function checks if the security parameters (including security level, compartment, transmission control code, and handling restrictions) in the incoming segment fit within the security ranges specified in the security list.

The data effects of this function are:

- Data examined only:
- ```
from_net.options [security]      sv.sec_ranges
```

MIL-STD-1778
12 August 1983

-- Return values

NO -- The values in the incoming segment are not within the ranges specified in the state vector.

YES -- The values in the incoming segment are within the ranges specified in the state vector.

9.4.6.2.16 Sec prec match? The sec_prec_match function compares the precedence level and security information (including security level, compartment, transmission control code, and handling restrictions) defined in the state vector against those of the incoming segment. The data effects of this function are:

a. Data examined only:
from NET.type_of_service.precedence sv.sec
from NET.options[security] sv.actual_prec

b. Return values:
NO -- The security and precedence of the segment do not match those of the state vector.

YES -- The security and precedence DO match.

if ((sv.sec = from NET.options[security]) and
(sv.actual_prec = from NET.type_of_service.precedence))

then return (YES)
else return (NO);

9.4.6.2.17 Seq# status? The seq#_status function compares the sequence number of the incoming segment against the current recv variables in the state vector to determine whether the segment contains data in the recv window. The data effects of this function are:

a. Data examined only:
from NET.seq_seq_num sv.recv_vndw
sv.recv_next

b. Return values:
VALID -- This segment does not contain data within the recv window.
INVALID -- This segment DOES contain data in the recv window.

--Due to zero length recv window and zero length segments,
--this decision function must examine four cases.
--These cases are expressed in the following conditional statements.

MIL-STD-1778
12 August 1983

```

if (from_NET.length = 0)
then if (sv.rcv_wndw = 0)
    then
        --When the segment contains no data, and the receive
        --window is closed, the segment sequence number
        --must equal the next expected to be acceptable.
        if (from_net.seq_num = sv.rcv_next)
            then return (VALID)
        else return (INVALID)
    else
        --When the segment contains no data and the receive
        --window is open, the segment sequence number must
        --fall within the receive window.
        if ((sv.rcv_next <= from_NET.seq_num) and
            (from_NET.seq_num < sv.rcv_next+sv.rcv_wndw))
            then return (VALID)
        else return (INVALID)
else if (sv.rcv_wndw = 0)
    then
        --When the segment carries data and the receive
        --window is closed, although no data can be
        --accepted, the control information is acceptable
        --if the segment sequence number exactly matches
        --the next expected.
        if (from_net.seq_num = sv.rcv_next)
            then return (VALID)
        else return (INVALID)
    else
        --When the segment carries data and the receive window
        --is open, the segment is acceptable if any data
        --falls within the receive window.
        if
            --Does the front of the data lie within the window?
            (((sv.rcv_next <= from_NET.seq_num)
              and (from_NET.seq_num <
                  sv.rcv_next+sv.rcv_wndw))
            or
            --Does the back of the data lie within the window?
            ((sv.rcv_next <= from_NET.seq_num+from_NET.
              length)
              and (from_NET.length+from_NET.seq_num <
                  sv.rcv_next+sv.rcv_wndw))
            or
            --Does the middle of the data lie within the window?
            ((sv.rcv_next > from_net.seq_num)
              and ( sv.rcv_next+sv.rcv_wndw <
                  (from_NET.length+from_NET.seq_num))))
            then return (VALID)
        else return (INVALID)

```

MIL-STD-1778
12 August 1983

9.4.6.2.18 SYN on? The SYN on function examines the SYN flag of the incoming segment. The data effects of this function are:

- a. Data examined only: from_NET.seg.syn_flag
 - b. Return values:
 - NO -- No SYN is present in the incoming segment.
 - YES -- A SYN is present in the segment.
- if (from_NET.seg.syn_flag = TRUE)
then return (YES)
else return (NO)

9.4.6.2.19 SYN in window? The SYN in window function determines whether an incoming segment contains a SYN, and if so, whether its sequence number lies in the recv window. The data effects of this function are:

- a. Data examined only:
 - from_NET.seg.syn_flag sv.recv_next
 - from_NET.seg.seq_num sv.recv_wndw
- b. Return values:
 - NO -- No SYN is present, or a SYN is present but it does not fall in the recv window.
 - YES -- A SYN is present and falls in the recv window.

--After a connection is established, no segments should contain
--SYNs. However, certain situations may produce a SYN.
--Shortly after a connection opens, a duplicate of the original
--SYN may arrive. It will not lie in the recv window, having
--already been accepted. Or, during a connection of long
--duration, very very rare error conditions may produce a SYN
with
--the recv window. This situation must be detected.

if ((from_NET.seg.syn_flag = TRUE) and
 (from_NET.seg.seq_num >= sv.recv_next) and
 (from_NET.seg.seq_num < sv.recv_next + sv.recv_wndw))
then return (YES)
else return (NO);

9.4.6.2.20 Zero recv window? The zero_recv_window function examines the recv variables to determine whether the recv window is zero, preventing the acceptance of any data from the remote TCP. The data effects of this function are:

- a. Data examined only: sv.recv_wndw

MIL-STD-1778
12 August 1983

- b. Return values:
NO -- The recv window is not zero. Data can be accepted.
YES -- The recv window IS zero. No data can be accepted.

```
if (sv.recv_wndw = 0)
then return (YES)
else return (NO);
```

9.4.6.3 Action procedures. The following action procedures represent the set of actions performed by a TCP entity state machine. They are called by the state and event correspondence defined in Section 9.4.6. These procedures have been organized and designed for clarity and are provided as guidelines. Although implementors can reorganize for better performance, the data effects of the resulting implementations must not differ from those specified below. Certain aspects of the actions described in the following procedures are subject to design choices. Specifically, the selection of strategies for handling retransmissions, sending acknowledgments, segmenting data, accepting data from the remote TCP, and delivering data to the ULP are governed by implementation dependent criteria. These strategies are encapsulated in "policy" procedures such as `accept_policy`. A policy procedure discusses the available approaches and returns information to an action procedure indicating appropriate processing. The policy procedures defined in the following section are: `accept_policy`, `ack_policy`, `deliver_policy`, `retransmit_policy`, and `send_policy`. The actions procedures invoke the execution environment primitives, defined in Section 10, to pass messages between protocol levels (TRANSFER), to read current time (CURRENT_TIME), and to set and cancel timers (SET_TIMER, CANCEL_TIMER).

9.4.6.3.1 Data management routines. This specification is intended to be as detailed and accurate as possible without implying a particular implementation approach or environment. However, a difficulty lies in the manipulation of internal data storage areas which is, by nature, implementation dependent. Thus, a set of data management routines are defined to manipulate the queues for send and receive data while specific data structures (such as arrays, linked lists, or circular buffers) remain undefined. The state vector can record send and receive variables in terms of sequence numbers because the data routines correlate sequence numbers to the physical position of data within the data structures. The data management routines defined in the following section are: `dm_add_to_recv`, `dm_add_to_send`, `dm_copy_from_send`, `dm_remove_from_send`, and `dm_remove_from_recv`.

9.4.6.3.2 Accept. The accept action procedure accepts data from the incoming segment and places it in the receive queue. The amount of data accepted is governed by the implementation dependent acceptance policy. An ACK is generated for the accepted data according to the ACK policy which is implementation dependent. Also, some data may be delivered according to implementation dependent delivery policies.

MIL-STD-1778
12 August 1983

The data effects of this procedure are:

- a. Data examined:
all fields in from_NET sv.recv_alloc
- b. Data modified:
all fields of to_NET sv.recv_wndw
sv.recv_next sv.recv_push
sv.recv_urg
- c. Local variables: start_seq amount offset

--The accept_policy procedure returns how much
--data is to be accepted, its beginning sequence number,
--and its location within the incoming segment.

accept_policy(amount, start_seq, offset);

if (amount > 0)
then

 dm_add_to_recv(start_seq, amount, offset);

--Update the recv_next sequence number if necessary.

 if (sv.recv_next = start_seq)
 then sv.recv_next := start_seq + amount;

--Record PUSH and URGENT information.

 if ((from_NET.seg.push_flag = TRUE) and
 (sv.recv_push < start_seq + amount))
 then sv.recv_push := start_seq + amount;

 if ((from_NET.seg.urg_flag = TRUE) and
 (sv.recv_urg < from_NET.seg.seq_num + from_NET.seg.urgptr))
 then sv.recv_urg := from_NET.seg.seq_num + from_NET.seg.urgptr;

--Refer to ack_policy to determine whether an ACK should be
generated

--at this point.

 to_ack := ack_policy();
 if (to_ack = TRUE)
 then send_ack(sv.recv_next);

--If the allocation allows, deliver data to the ULP.

 if (sv.recv_alloc > 0)
 then deliver;
end;

9.4.6.3.3 Accept policy. As one of the policy procedures, accept_policy discusses the alternative strategies for accepting the data of incoming segments and returns to the calling procedure the number of data octets to be accepted. The parameters are:

MIL-STD-1778
12 August 1983

- a. starting_seq - sequence number of the first octet of data to be accepted
- b. quantity - the number of octets of data to be accepted
- c. segment_data_offset - the position of the first data octet within the incoming segment's text to be accepted.

9.4.6.3.4 Accept strategy. A TCP implementation may use one of several strategies to accept data within the receive window from an incoming segment.

- a. Accept in-order data only. The acceptance test is:

$$\text{from_NET.seq.seq_num} = \text{sv.recv_next}$$
 That is, the sequence number of the incoming segment must exactly equal the next sequence number expected to be received.
- b. Accept any data within the receive window. The acceptance test has several parts:

$$\text{sv.recv_next} \leq \text{from_NET.seq.seq_num}$$

$$\leq \text{sv.recv_next} + \text{sv.recv_wndw}$$

- or -

$$\text{sv.recv_next} \leq \text{from_NET.seq.seq_num} + \text{length}$$

$$\leq \text{sv.recv_next} + \text{sv.recv_wndw}$$

That is, any portion of the text falling within the receive window (i.e., in the interval between the next sequence number expected to be received and the last sequence number in the window) is accepted.

9.4.6.3.5 "In-order" strategy. The "in-order" strategy allows a simple acceptance test and a straightforward scheme for data storage. However, the loss of a single segment can result in the remote TCP retransmitting every succeeding segment. The "in-the-window" strategy requires a more involved acceptance test and a sophisticated data storage scheme to keep track of data accepted out of order. Also, as each segment is accepted, the data storage must be checked so that a contiguous interval of out-of-order data can be recognized. This strategy allows the remote TCP to retransmit only lost segments.

9.4.6.3.6 Ack policy. As one of the policy procedures, ack_policy discusses the alternative strategies for acknowledging data accepted from incoming segments and returns to the calling action procedure a boolean value indicating whether an acknowledgment should be sent. A TCP implementation may apply one of two acknowledgment timing schemes.

- a. When data is accepted from remote TCP, immediately generate an empty segment containing current acknowledgment information and return it to the remote TCP.

MIL-STD-1778
12 August 1983

- b. When data is accepted from remote TCP, record the need to acknowledge data in the state vector, but wait for an outbound segment with data on which to piggyback the ACK. However to avoid a long delay, set an "ack timer" to limit the delay to a reasonable interval. Thus, if no outbound segment with data is produced within the chosen ack timeout interval, the timer expires and an empty ACK segment is generated and sent to the remote TCP. If a data segment is produced before the timer expires, the timer is cancelled and the need to acknowledge record is erased from the state vector. (Note that no "ack timeout" event appears in this standard. This event and the resulting call to the `send_ack` action procedure should be added if the this approach is taken.)

The trade-off between the two approaches is processing time versus control overhead. The "automatic" ack approach is simple, but results in extra segment generation. The "timed" ack approach requires more processing but will reduce the number of segments generated on connections with two-way data transfer.

9.4.6.3.7 Check urg. The `check_urg` action procedure examines the header of the incoming segment to determine whether new urgent information is present. If so, the urgent pointer is recorded in the `recv` variables. The data effects of this procedure are:

```

a. Data examined:
    from_NET.seg.urg_flag    from_NET.seg.data_offset
    from_NET.seg.urgptr      from_NET.length
    from_NET.seg.seq_num

b. Data modified:  sv.recv_urg

begin
--Check urgent flag and urgent pointer.
  if (from_NET.seg.urg_flag = TRUE)

  then --Check to see if a new urgent pointer is present.
    if (sv.recv_urg < from_NET.seg.seq_num + from_NET.seg.urgptr)
    then
      if (sv.recv_urg < sv.recv_save)
      then --If the ULP is not in "urgent mode", it must be
        --informed of the presence of urgent information.
        --implementation dependent action
        sv.recv_urg := from_NET.seg.seq_num + from_NET.seg.urgptr;
    end;
end;
```

9.4.6.3.8 Compute checksum. The `compute_checksum` procedure computes the checksum of an outbound segment and places the value in the header's checksum field.

The data effects of this function are:

MIL-STD-1778
12 August 1983

- a. Data examined
 all fields of to_NET.seg to_NET.protocol
 to_NET.source_addr to_NET.length
 to_NET.destination_addr

- b. Data modified: to_NET.seg.checksum

begin

--The checksum algorithm is the 16-bit one's complement of the
 --one's complement sum of all 16-bit words in the segment
 --header and segment text. If a segment contains an odd number
 --of octets, the last octet is padded on the right with zeros
 --to form a 16-bit word for checksum purposes. While computing
 --the checksum, the checksum field itself is replaced with zeros.
 --The checksum includes a 96-bit pseudo header prefixed to the
 --actual TCP header. As, the diagram shows, the pseudo header
 --contains the source address, the destination address, the
 --protocol identifier, and the length of the TCP segment
 --(not counting the pseudo header).

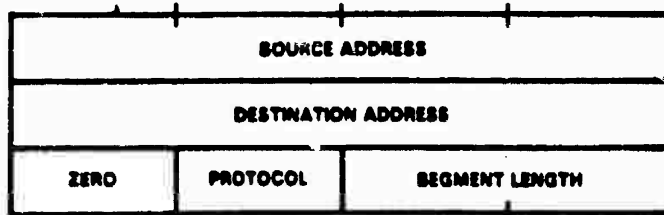


FIGURE 15. Compute checksum procedure.

--The actual computation is implementation dependent.
 end;

9.4.6.3.9 Conn open. The conn_open action procedure is called just before a connection enters the ESTABLISHED state. According to the implementation policy, the procedure updates the ACK and window information, delivers any waiting data, and acknowledges any received data. The ULP is notified of the newly opened connection with an OPEN_SUCCESS service response. The data effects of the procedure are:

- Data examined: sv.lcn
- Data modified: to_ULP.service_response to_ULP.lcn
- Local variables: delivery_amount need_to_ack

begin

--Inform the ULP.
 to_ULP.service_response := OPEN_SUCCESS;
 to_ULP.lcn := sv.lcn;
 TRANSFER to_ULP to the ULP identified by sv.source_port;

MIL-STD-1778
12 August 1983

```

--The incoming segment contained either a SYN and an ACK of
--our SYN, or just an ACK. In either case, use the new
--ACK to update the send variables.
    update;

--Based on implementation dependent policies, deliver any waiting
--data to the ULP.
    delivery_amount := deliver_policy();
    if (delivery_amount > 0) then deliver;

--Based on implementation dependent acking policy, ack the
--incoming segment.
    need_to_ack := ack_policy();
    if (need_to_ack = TRUE) then send_ack;
end;
```

9.4.6.3.10 Deliver. The deliver action procedure moves data accepted from the remote TCP into the to_ULP structure for delivery to the ULP. The amount of data delivered is based on the receive allocation, the amount of pushed data, and the implementation dependent delivery policy. The data effects of this procedure are:

- a. Data examined:

sv.recv_push	sv.recv_next
sv.recv_urg	sv.recv_finflag
sv.lcn	sv.source_port
- b. Data modified:

sv.recv_save	all fields of to_ULP
sv.recv_alloc	
- c. Local variables: pushed delivery_amount urgent_length

```

begin
--Does pushed data await delivery?
if (sv.recv_push > sv.recv_save)
then --Pushed data waits so compute amount needing delivery.
    if (sv.recv_push > sv.recv_next)
    then pushed := sv.recv_next - sv.recv_save
    else pushed := sv.recv_push - sv.recv_save;

    --Is there enough allocation for all the pushed data?
    if (sv.recv_alloc < pushed)
    then delivery_amount := sv.recv_alloc
    else delivery_amount := pushed;

else --No pushed data waits. Refer to the deliver_policy
--to determine how much data should be passed to the
--ULP at this point.
    delivery_amount := deliver_policy();

--Deliver computed amount of data to ULP, including urgent
--information.
```

MIL-STD-1778
12 August 1983

```
if (delivery_amount > 0)
then begin
  --Check for "end of urgent" in data which cannot be delivered
  --in the same delivery unit with subsequent non-urgent data.

  if ((sv.recv_urg > sv.recv_save) and
      (sv.recv_urg < sv.recv_save + delivery_amount))

  then --Deliver the urgent data alone first.
  begin
    urgent_length := sv.recv_urg - sv.recv_save;

    dm_remove_from_recv(sv.recv_save, urgent_length);
    to_ULP.data_length := urgent_length;
    --Note that implementation dependent delivery unit
    --size restrictions are not handled.
    to_ULP.urgent_flag := TRUE;
    to_ULP.lcn := sv.lcn;
    to_ULP.service_response := DELIVER;
    TRANSFER to_ULP to the ULP named by to_ULP.source_port;

    sv.recv_save := sv.recv_urg;
    sv.recv_alloc := sv.recv_alloc - urgent_length;
    delivery_amount := delivery_amount - urgent_length;
  end;

  --Move data without an end of urgent into to_ULP.data
  --and deliver to ULP.
  dm_remove_from_recv(sv.recv_save, delivery_amount);
  to_ULP.data_length := delivery_amount;
  --Note that implementation dependent delivery unit
  --size restrictions are not handled.
  to_ULP.lcn := sv.lcn;
  if (sv.recv_save < sv.recv_urg)
  then to_ULP.urgent_flag := TRUE
  else to_ULP.urgent_flag := FALSE;

  TRANSFER to_ULP to the ULP named by sv.source_port;

  --Update recv variables.
  sv.recv_save := sv.recv_save + delivery_amount;
  sv.recv_alloc := sv.recv_alloc - delivery_amount;

  --If the remote side has closed, and this data clears the
  --receive queue, the ULP must be notified.
  if ((sv.recv_finflag = TRUE) and
      (sv.recv_next = sv.recv_save))
```

MIL-STD-1778
12 August 1983

```

    then
      begin
        to_ULP.service_response := CLOSING;
        to_ULP.lcn := sv.lcn;
        TRANSFER to_ULP to the ULP named by sv.source_port;
      end;

    end; --of data delivery to ULP
  end;
end;

```

9.4.6.3.11 Deliver policy. As one of the policy procedures, deliver_policy discusses the alternative strategies for delivering data to the ULP. It returns to the calling procedure the number of octets of data to be delivered. Barring zero receive allocations and pushed data, the TCP specification allows an implementation to deliver data to the ULP at its own convenience. However, performance considerations should be examined. On one hand, data available for delivery should be delivered with reasonable promptness; it should not be delayed indefinitely while waiting for a delivery units worth to arrive. On the other hand, the data should not be delivered an octet at a time wasting both internal TCP processing time and external execution environment resources. A reasonable compromise can be achieved guided by system design criteria.

9.4.6.3.12 Dispatch. The dispatch action procedure accepts the data and interface parameters passed by the ULP in a send request, adds the data to the send queue, and adjusts appropriate send variables. Depending on the send policy, the procedure may segment and transmit some portion of data to the remote TCP. The data effects of this procedure are:

- a. Data examined:

from_ULP.lcn	from_ULP.push_flag
from_ULP.data	from_ULP.urgent_flag
from_ULP.data_length	from_ULP.ulp_timeout
- b. Data modified:

sv.send_free	sv.send_next
sv.ulp_timeout	sv.send_una
sv.send_push	sv.send_wndw
sv.send_urg	

```

begin
  --Save the data along with timestamp, starting at sv.send_free,
  --then update the send variables.

  add_to_send(sv.send_free, from_ULP.data_length, CURRENT_TIME());

  sv.send_free := sv.send_free + from_ULP.length;

  if (from_ULP.push_flag = TRUE)
  then sv.send_push := sv.send_free;

```

MIL-STD-1778
12 August 1983

```

    if (from_ULP.urgent_flag = TRUE)
    then sv.send_urg := sv.send_free;

    --Depending on implementation, the ULP timeout timer may
    --need to be restarted when the interval is changed by the ULP.
    if ((from_ULP.ulp_timeout /= NULL)
    --option exercised to set timeout and (from_ULP.ulp_timeout /=
    sv.ulp_timeout)) then sv.ulp_timeout := from_ULP.ulp_timeout;

    --Call the send_new_data procedure to determine if any
    --newly received data can be sent at this time.

    send_new_data;

    end; --non-zero send window processing
end;
```

9.4.6.3.13 Dm add to send. As one of the data management routines, the dm_add_to_send procedure adds the data provided by the ULP in from_ULP.data to the send storage area. The calling sequence is:

```

dm_add_to_send( seq_num, length, time )

seq_num = the sequence number of the first octet
         being added to the storage area

length = the number of octets to be added

time = the current time to be associated with each
      data octet for later determination of data age
      for the ULP timeout.
```

9.4.6.3.14 Dm add to recv. As one of the data management routines, the dm_add_to_recv procedure copies data from an incoming segment, found in from_NET.seg.data, into the receive storage area. This routine is called as segments are validated and portions of their data are found to be in the receive window. The calling sequence is:

```

dm_add_to_recv( seq_num, length, offset )

seq_num = the sequence number of the first octet to be copied.

length = the number of data octet to be copied.

offset = the location of first octet to be taken from the
        data portion of the segment.
```

9.4.6.3.15 Dm copy from send. As one of the data management routines, the dm_copy_from_send procedure copies data from the send storage area into to_NET.seg.data. This routine is used as data is segmented and transmitted initially, and as retransmissions are required. The calling sequence is:

MIL-STD-1778
12 August 1983

`dm_copy_from_send(seq_num, length)`

`seq_num` = the sequence number of the first data octet to be copied into `to_NET.req.data`

`length` = the number of octets to be copied

9.4.6.3.16 Dm remove from rcv. As one of the data management routines, the `dm_remove_from_rcv` routine removes data from the receive storage area and places it in the `to_ULP.data` structure. This is called as data is delivered to the ULP. The calling sequence is:

`dm_remove_from_rcv(seq_num, length)`

`seq_num` = the sequence number of the first octet to be removed and copied

`length` = the number of data octets to be removed and copied

9.4.6.3.17 Dm remove from send. As one of the data management routines, the `dm_remove_from_send` procedure deletes data from the send storage area. This routine is called as data is acknowledged by the remote TCP and removed from the retransmission "queue." The calling sequence is:

`dm_remove_from_send(seq_num, length)`

`seq_num` = the sequence number of the first octet to be removed.

`length` = the number of data octets to be removed.

9.4.6.3.18 Error. The error procedure fills in the fields of `to_ULP` with the local connection name, the ERROR service response, and the error description passed by parameter. This information is passed to the local ULP.

a. Data examined: `sv.lcn` `sv.source_port`

b. Data modified: `to_ULP.lcn` `to_ULP.service_response`
 `to_ULP.error_desc`

begin

 --Construct an error message for the local ULP.

`to_ULP.service_response := ERROR;`

`to_ULP.lcn := sv.lcn;`

`to_ULP.error_desc := parameter;`

 TRANSFER `to_ULP` to the ULP named by `sv.source_port`;

end;

MIL-STD-1778
12 August 1983

9.4.6.3.19 Format net params. The format_net_params procedure fills in the parameters used by the network protocol entity after the calling procedure has filled in the outgoing segment header. The size of the segment's text portion is passed by parameter.

```

a. Data examined:
   to_NET.seg.data_offset
   sv.destination_addr
   sv.destination_port
   sv.source_addr
   sv.source_port

b. Data modified:
   to_NET.identifier
   to_NET.protocol
   to_NET.destination_addr
   to_NET.seg.source_port
   to_NET.dont_fragment
   to_NET.type_of_service
   to_NET.length
   to_NET.source_addr
   to_NET.destination_port

begin
  --Fill in the network parameters.
  to_NET.seg.source_port := sv.source_port;
  to_NET.seg.destination_port := sv.destination_port;
  to_NET.source_addr := sv.source_addr;
  to_NET.destination_addr := sv.destination_addr;
  to_NET.protocol := TCP_ID;
  to_NET.type_of_service.precedence := sv.actual_prec;
  to_NET.type_of_service.reliability := NORMAL;
  to_NET.type_of_service.delay := NORMAL;
  to_NET.type_of_service.throughput := NORMAL;
  to_NET.identifier := gen_id();
  to_NET.dont_fragment := FALSE;
  to_NET.time_to_live := ONE_MINUTE_TTL;
  to_NET.length := to_NET.seg.data_offset +
    parameter;
  to_NET.options[security] := sv.sec;
end;
```

9.4.6.3.20 Gen id. The gen_id action procedure returns an identifier to the calling procedure to be passed to the network protocol entity when a segment is transmitted with a NET_SEND primitive.

```

a. Data examined: --implementation dependent

b. Data modified: --none

begin
  --The generation of the identifier is implementation dependent.
  --The network protocol entity uses the identifier, along with
  --addressing information, to distinguish between sending units
  --(i.e. datagrams) if fragmentation and reassembly are required.
  --So, TCP must generate unique identifiers for each segment if
  --the data is to be transmitted without confusion.
  --
```

MIL-STD-1778
12 August 1983

- Also, if a retransmitted segment is accompanied by the
- identifier used for its original transmission, the network
- protocol entity may be able to piece together parts of the
- original and the retransmission to improve its performance.
- Note that if repackaging is performed during retransmission,
- the original identifier cannot be used.

end;

9.4.6.3.21 Gen_isn. The gen_isn procedure returns an initial sequence number to the calling procedure for use during the three-way handshake of connection establishment.

a. Data examined: --implementation dependent

b. Data modified: - none -

--implementation dependent action

9.4.6.3.22 Gen_lcn. The gen_lcn procedure returns a local connection name, or lcn, to the calling procedure to be used as a shorthand identifier by TCP and the local ULP in service requests and responses pertaining to a connection.

a. Data examined: --implementation dependent

Data modified: - none -

begin

- The generation of the lcn is implementation dependent.
- A TCP entity usually supports many connections.
- If the lcn is a pointer or table index, service requests
- can be quickly matched to their state vector.

end;

9.4.6.3.23 Gen_syn. The gen_syn action procedure formats and transmits a segment containing a SYN to the remote TCP. As part of the SYN generation, an initial sequence number is selected. The procedure accepts one parameter whose values are ALONE, WITH_ACK, and WITH_DATA, indicating whether the segment will contain an ACK or data. This procedure does not handle generating a SYN carrying a FIN flag because the specified service interface does not support a transaction primitive described in Appendix C. However, if such primitive were created, this procedure would have to be modified to handle it. The data effects of this procedure are:

a. Data examined:

sv.source_port	sv.recv_isn
sv.source_addr	sv.recv_next
sv.destination_port	sv.send_next
sv.destination_addr	sv.send_free
sv.recv_wndw	sv.send_push
sv.send_urg	

MIL-STD-1778
12 August 1983

b. Data modified:
 sv.send_isn sv.send_next
 all fields of to_NET sv.send_una

c. Local variables: amount

```
begin
  --Generate the initial sequence number to be used for
  --data sent to the remote TCP.
  sv.send_isn := gen_isn();
  sv.send_next := sv.send_isn + 1; --SYN uses the first seq#.
  sv.send_una := sv.send_isn;

  to_NET.seq.seq_num := sv.send_isn;

  --Check parameter to determine exact type of SYN.
  case parameter of

    when ALONE =>
      to_NET.seq.ack_flag := FALSE;
      to_NET.seq.wndw     := 0;
      to_NET.seq.push_flag := FALSE;
      to_NET.seq.urg_flag := FALSE;
      amount := 0;

    when WITH_ACK =>
      to_NET.seq.ack_flag := TRUE;
      to_NET.seq.wndw     := sv.recv_wndw;
      to_NET.seq.ack_num := sv.recv_isn + 1;
      to_NET.seq.push_flag := FALSE;
      to_NET.seq.urg_flag := FALSE;
      amount := 0;

    when WITH_DATA =>
      to_NET.seq.ack_flag := FALSE;
      to_NET.seq.wndw     := 0;

      --The data supplied by the ULP is in the send queue.
      --However, the amount of data to accompany the SYN
      --is determined by the send_policy.

      amount := send_policy();

      if (amount > 0)
      then dm_copy_from_send( sv.send_next, amount );
        if (sv.send_push = sv.send_next + amount)
        then to_NET.seq.push_flag := TRUE;
        sv.send_next := sv.send_next + amount;
        else to_NET.seq.push_flag := FALSE;
      end if;
  end case;
```

MIL-STD-1778
12 August 1983

```
--Add the urgent information regardless of data length.
if (sv.send_urg >= to_NET.seq_num)
then to_NET.seq.urg_flag := TRUE;
  to_NET.seq.urgptr := sv.send_urg -
  to_NET.seq.seq_num;
else to_NET.seq.urg_flag := FALSE;

if (MAX_SEGMENT_SIZE option used in this implementation)
then
  to_NET.seq.options[1] := 2;    --Max header size
                                option kind
  to_NET.seq.options[2] := 4;    --option length =
                                4 octets
  to_NET.seq.options[3..4] := MAX_SEGMENT_SIZE;
                                --impl.dep value
  to_NET.seq.data_offset := 6;
else
  to_NET.seq.data_offset := OPTIONLESS_HEADER;

format_net_params(amount);
compute_checksum;
TRANSFER to_NET to the network protocol entity.
end;
```

9.4.6.3.24 Load security. The security parameters (including security level, compartment, transmission control code, and handling restrictions) in an incoming segment are loaded into the state vector.

The data effects of this function are:

- Data examined:
 - from_net_options [security]
 - Data modified:
 - sv.sec
- This would only occur after a successful sec_range_match.
sv.sec := from_net_options [security]

9.4.6.3.25 New allocation. The new_allocation action procedure takes the new value provided by the ULP in an allocation service request and adds it to the current receive allocation. Data waiting for this allocation is delivered to the ULP. The data effects of this procedure are:

- a. Data examined: from_ULP.data_length
- b. Data modified: sv.recv_alloc

MIL-STD-1779
12 August 1983

```
begin
  --Add in the new receive allocation.
  sv.recv_alloc := sv.recv_alloc + from_ULP.data_length;
  --Depending on implementation dependent window management strategy,
  --this new receive allocation may be factored into a new
  --value for the receive window.

  --If data awaits this allocation, deliver it.
  deliver;
end;
```

9.4.6.3.26 Open. The open action procedure records the parameters from an open service request (either Active Open, Fully Specified Passive Open, or Unspecified Passive Open), assigns a local connection name, and returns it to the ULP in an OPEN_ID service response. The data effects of this procedure are:

```
a. Data examined:
   from_ULP.request_name
   from_ULP.source_port
   from_ULP.destination_port
   from_ULP.destination_addr
   from_ULP.precedence
   from_ULP.security
   from_ULP.sec_ranges
   from_ULP.timeout
   from_ULP.timeout_action

b. Data modified:
   sv.source_port
   sv.source_addr
   sv.destination_port
   sv.destination_addr
   sv.lcn
   sv.original_prec
   sv.sec, sv.sec_ranges
   sv.ulp_timeout, sv.ULP_timeout_
   action
   to_ULP.service_response
   to_ULP.source_port
   to_ULP.source_addr
   to_ULP.destination_addr
   to_ULP.destination_port
   to_ULP.lcn
```

```
begin
  --Assign a local connection name according to
  --implementation dependent algorithms.
  sv.lcn := gen_lcn();

  --The security, precedence, and timeout parameters are
  --optional. If they are not provided by the ULP, default
  --values are assigned. For security and precedence defaults
  --in nonsecure environments, the lowest levels are generally used.
  --A timeout default is more arbitrary, but the current
  --suggested value is two minutes.

  if (from_ULP.security is present)
  then sv.sec := from_ULP.security
  else sv.sec := DEFAULT_SECURITY;
```

MIL-STD-1778
12 August 1983

```
if (from_ULP.precendence is present)
then sv.original_prec := from_ULP.precendence;
   sv.actual_prec := from_ULP.precendence;
else sv.original_prec := DEFAULT_PRECEDENCE;
   sv.actual_prec := DEFAULT_PRECEDENCE;

if (from_ULP.timeout is present)
then sv.ulp_timeout := from_ULP.timeout
else sv.ulp_timeout := DEFAULT_TIMEOUT;

if (from_ULP>timeout_action is present
then sv.ULP_timeout_action := from_ULP.timeout_action
else sv.ULP_timeout_action := DEFAULT_TIMEOUT_action

--The source port is provided in all open requests. The source
--address is the address of this TCP entity.
sv.source_port := from_ULP.source_port;
sv.source_addr := THIS_ADDRESS;
--The remaining parameters vary according to open request type.

case from_ULP.request_name of

when Unspecified_Passive_Open =>
   --This request does not carry the destination
   --socket. It remains unassigned until a matching
   --SYN from a remote TCP arrives.
   sv.open_mode := PASSIVE;
   sv.sec_ranges := from_ULP.sec_ranges;

when Full_Passive_Open =>
   sv.destination_addr := from_ULP.destination_addr;
   sv.destination_port := from_ULP.destination_port;
   sv.open_mode := PASSIVE;
   sv.sec_ranges := from_ULP.sec_ranges;

when Active_Open =>
   sv.destination_addr := from_ULP.destination_addr;
   sv.destination_port := from_ULP.destination_port;
   sv.open_mode := ACTIVE;

when Active_Open_With_Data =>
   sv.destination_addr := from_ULP.destination_addr;
   sv.destination_port := from_ULP.destination_port;
   sv.open_mode := ACTIVE;

   --Record data accompanying open request.
   save_send_data;

end case;
```

MIL-STD-1778
12 August 1983

--Return the local connection name assigned.

```
to_ULP.service_response := OPEN_ID;
to_ULP.source_port := sv.source_port;
to_ULP.source_addr := sv.source_addr;
to_ULP.destination_port := sv.destination_port;
to_ULP.destination_addr := sv.destination_addr;
to_ULP.lcn := sv.lcn;
```

TRANSFER to_ULP to the ULP named by sv.source_port;

end;

9.4.6.3.27 Openfail. The openfail action procedure informs the ULP that the attempted connection could not be opened. It also clears the state vector. The data effects of the procedure are:

a. Data examined: sv.lcn sv.source_port

b. Data modified:

```
all state vector elements
to_ULP.lcn to_ULP.service_response
```

--Construct an OPEN_FAIL message for the ULP.

```
to_ULP.service_response := OPEN_FAIL;
```

```
to_ULP.lcn := sv.lcn;
```

TRANSFER to_ULP to the ULP named by sv.source_port;

--The state vector is cleared without generating a ULP message.

```
reset_self(NO_REPORT);
```

9.4.6.3.28 Part reset. The part_reset action procedure clears the send and recv variables without terminating the connection. The data effects of the procedure are:

a. Data examined: sv.open_mode

b. Data modified: all send and receive variables

begin

--The remote TCP address and port are cleared if the connection

--open mode was PASSIVE.

```
if (sv.open_mode = PASSIVE)
```

```
then
```

```
sv.destination_port := NULL;
```

```
sv.destination_addr := NULL;
```

--Clear all variables set during the connection opening

--handshake.

```
do_remove_from_send(sv.send_una, QUEUE_SIZE);
```

```
do_remove_from_recv(sv.recv_free, QUEUE_SIZE);
```

MIL-STD-1778
12 August 1983

```

sv.actual_prec    := NULL;    sv.send_next      := NULL;
sv.recv_isn       := NULL;    sv.send_una       := NULL;
sv.recv_next      := NULL;    sv.send_wndw      := NULL;
sv.recv_wndw      := NULL;    sv.send_push      := NULL;
sv.recv_alloc     := NULL;    sv.send_urg       := NULL;
sv.recv_push      := NULL;    sv.send_finflag   := NULL;
sv.recv_urg       := NULL;    sv.send_free      := NULL;
sv.recv_save      := NULL;    sv.send_lastup1   := NULL;
sv.recv_finflag   := NULL;    sv.send_lastup2   := NULL;
sv.send_isn       := NULL;    sv.send_max_seg   := NULL;

```

end;

9.4.6.3.29 Raise prec. The raise precedence action procedure raises the precedence level recorded in the state vector to the level provided by the remote TCP. paragraph 9.2.11 of the entity overview discusses precedence negotiation during connection establishment. The data effects of this procedure are:

- a. Data examined: from_NET.type_of_service.precedence
- b. Data modified: sv.actual_prec
 - A SYN from the remote TCP carries a precedence level
 - greater than that indicated by the local ULP.
 - Precedence is carried as a type of service parameter.
 - sv.actual_prec := from_NET.type_of_service.precedence;

9.4.6.3.30 Record syn. The record_syn action procedure records the control information from the incoming segment containing a SYN flag. The data effects of this procedure are:

- a. Data examined: all fields of from_NET
- b. Data modified:

sv.recv_next	sv.send_wndw
sv.recv_urg	sv.send_una
sv.recv_isn	sv.destination_port
sv.send_max_seg	sv.destination_addr
sv.recv_push	
- c. Local variables: start_seq amount offset

begin

```

--If this half of the connection was opened passively, the
--remote information should be added to the state vector.
if (sv.open_mode = PASSIVE)
then

```

```

sv.destination_port := from_NET.seg.source_port;
sv.destination_addr := from_NET.source_addr;

```

MIL-STD-1778
12 August 1983

```

--Record rcv_data.
sv.rcv_isn := from_NET.seq_num;
sv.rcv_next := sv.rcv_isn+1;

--Record send data.
if (from_NET.seq.ack_flag = TRUE)
then sv.send_una := from_NET.seq.ack_num;

--Record maximum segment size if present in option field.
if ((from_NET.seq.data_offset > 5)      --optionless header size
and (from_NET.seq.options[0] = 2)) --Max Seg Option Kind
then
sv.send_max_seg := from_NET.seq.option[3..4];

--If data accompanied the SYN, apply the implementation
--dependent data acceptance policy to determine how much
--data should be saved, its position in the rcv_queue,
--and its position in the incoming segment.

accept_policy( start_seq, amount, offset );

if (amount > 0)

then
add_to_rcv( start_seq, amount, offset );

--Update the rcv_next sequence number if necessary.
if (sv.rcv_next = start_seq)
then sv.rcv_next := start_seq + amount;
else --record data position in receive storage area
--implementation dependent action
--Record PUSH and URGENT information.
if ((from_NET.seq.push_flag = TRUE) and
(sv.rcv_push < start_seq + amount))
then sv.rcv_push := start_seq + amount;

if ((from_NET.seq.urg_flag = TRUE) and
(sv.rcv_urg < from_NET.seq.seq_num + from_NET.seq.urgptr))
then --record the new urgent data position
sv.rcv_urg := from_NET.seq.seq_num + from_NET.seq.urgptr;

end;

```

9.4.6.3.31 Report timeout (sv *). The report_timeout action procedure informs the ULP that a ULP_timeout has occurred. The oldest data in the send queue is requested and the timeout time reset.

The data effects of this function are:

- Data examined:
- Data modified:

MIL-STD-1778
12 August 1983

```

begin
    error(sv_.lcn,"ULP_timeout)
    transfer to ULP to the ULP names by sv_._source_port;
    requeue_oldest (sv_*);
end;

```

9.4.6.3.32 Requeue oldest (sv *). The requeue_oldest action procedure removes the oldest data from the send_queue and requeues the data making it the youngest.

The data effects of this procedure are:

- Data examined:

sv.*send_queue

9.4.6.3.33 Reset. The reset action procedure formats and sends a segment with a reset flag to the remote TCP to terminate the connection. RESET segments must be formatted so that the remote TCP finds the segments acceptable. The procedure accepts one parameter indicating the format of the RESET segment to be sent. The parameter value "SEG" indicates that the incoming segment determines the format. If the segment contains an ACK, this forms the basis of the sequence number in the RESET segment. If the segment does not contain an ACK, the RESET segment is made acceptable by carrying an ACK of the incoming segment's text. The parameter value CURRENT indicates that the RESET is not the result of an incoming segment, but because of a ULP abort request or the ULP timeout. In such situations, the RESET segment is formed with a sequence number based on current state vector values. The data effects of this procedure are:

a. Data examined:

sv. <u>source_port</u>	sv. <u>sec</u>
sv. <u>source_addr</u>	sv. <u>actual_prec</u>
sv. <u>destination_port</u>	sv. <u>send_next</u>
sv. <u>destination_addr</u>	sv. <u>recv_next</u>

b. Data modified: -none-

```

begin
    --Based on the parameter, set the sequence and ack numbers.
    if (parameter = SEG)
    then --Check the incoming segment for ACK presence.
        if (from_NET.seg.ack_flag = TRUE)
        then
            to_NET.seg.seq_num := from_NET.seg.ack_num;
            to_NET.seg.ack_flag := FALSE;
        else
            to_NET.seg.seq_num := 0;
            to_NET.seg.ack_flag := TRUE;
            to_NET.seg.ack_num := from_NET.seg.seq_num +
                (from_NET.length - from_NET.seg.data_offset*4);

```


MIL-STD-1778
12 August 1983

```

else --parameter = CURRENT, so use current state vector values.
  to_NET.seq.seq_num      := sv.send_next;
  to_NET.seq.ack_flag     := FALSE;

--Form a segment using current state vector data, set the
--reset flag, and transmit to the remote TCP.
  to_NET.seq.rst_flag     := TRUE;
  to_NET.seq.syn_flag     := FALSE;
  to_NET.seq.urg_flag     := FALSE;
  to_NET.seq.push_flag    := FALSE;
  to_NET.seq.fin_flag     := FALSE;
  to_NET.seq.window       := 0;
  to_NET.seq.data_offset  := OPTIONLESS_HEADER;

  format_net_params( 0 );
  compute_checksum;
  TRANSFER to_NET to the network protocol entity;

end;
```

9.4.6.3.34 Reset self. The reset self action procedure informs the ULP that the connection is terminating, and then sets the state vector elements to their initial values. The reset self procedure has one parameter indicating the reason for connection termination. If the parameter equals NO_REPORT, no service response is prepared for the ULP. All other values produce service responses including RR for remote reset, NF for network failure, UT for ULP timeout, SP for security or precedence mismatch, UC for user close, and UA for user abort. The data effects of this procedure are:

- a. Data examined: sv.lcn
- b. Data modified: all state vector elements

```

begin
if parameter /= NO_REPORT
then begin
  case parameter of
    when RA =>
      to_ULP.error_desc := "Remote abort."
    when NF =>
      to_ULP.error_desc := "Network failure."
    when SP =>
      to_ULP.error_desc := "Security/precedence mismatch."
    when UT =>
      to_ULP.error_desc := "ULP timeout."
    when UA =>
      to_ULP.error_desc := "ULP abort."
    when UC =>
      to_ULP.error_desc := "ULP close."
    when SF =>
      to_ULP.error_desc := "Service failure."
  end case;
end;
```

MIL-STD-1778
12 August 1983

```
to_ULP.lcn := sv.lcn;
to_ULP.service_response := TERMINATE;
TRANSFER to_ULP to the ULP identified by sv.source_port;
end;
```

--Regardless of the cause, clear all queues and initialize state vector.

```
part_reset;
sv.source_port := NULL;
sv.source_addr := NULL;
sv.destination_port := NULL;
sv.destination_addr := NULL;
end;
sv.lcn := NULL;
sv.sec := NULL;
sv.original_prec := NULL;
sv.actual_prec := NULL;
sv.ulp_timeout := NULL;
```

9.4.6.3.35 Restart time wait. The restart_time_wait action procedure restarts the currently running "time wait" timer. This procedure is called after a retransmitted FIN is seen from the remote TCP. The data effects of this procedure are:

- a. Data examined: - none -
- b. Data modified: - none -

```
--Cancel the existing timer and start it up from scratch.
cancel_timer( TIME_WAIT, sv.lcn );
start_timer( TIME_WAIT, sv.lcn, TIME_WAIT_INTERVAL );
```

9.4.6.3.36 Retransmit. The retransmit actions procedure resends data that has not been acknowledged within the retransmission timeout interval. Because the amount of data resent is implementation dependent, this decision is encapsulated in the retransmit_policy procedure. The data effects of this procedure are:

- a. Data examined:

sv.send_una	sv.send_wndw
sv.send_next	sv.send_max_seg
sv.send_push	sv.send_urg
sv.send_finflag	sv.send_free
sv.recv_next	sv.recv_wndw

- b. Data modified:
 - all fields of to_NET
 - retransmission timer

- c. Local variables: retrans_amount start_pt pushed_amount

```
begin
--Determine how much data should be retransmitted to the
--remote TCP.
retrans_amount := retransmit_policy();
```

MIL-STD-1778
12 August 1983

```

if (retrans_amount > 0)
then
begin
  --Starting from the front of the retransmission queue,
  --segment and retransmit data indicated by amount.

  start_pt := sv.send_una;
  to_NET.seg.seq_num := start_pt;
  to_NET.seg.rst_flag := FALSE;

  if (start_pt = sv.send_isn)
  then --The SYN is being retransmitted.

  to_NET.seg.syn_flag := TRUE;
  if (sv.rcv_isn = NULL) --Has the remote TCP been heard from?
  then to_NET.seg.ack_flag := FALSE;
    to_NET.seg.wndw := 0;
  else to_NET.seg.ack_num := sv.rcv_next;
    to_NET.seg.ack_flag := TRUE;
    to_NET.seg.wndw := sv.rcv_wndw;

  if (MAX_SEGMENT_SIZE option used in this implementation)
  then
    to_NET.seg.options[1] := 2; --See section 6.2.11
    to_NET.seg.options[2] := 4; --for option format.
    to_NET.seg.options[3..4] := MAX_SEGMENT_SIZE;
    to_NET.seg.data_offset := 6;
  else to_NET.seg.data_offset := OPTIONLESS_HEADER;

  else --Normal data retransmission.
    to_NET.seg.ack_num := sv.rcv_next;
    to_NET.seg.ack_flag := TRUE;
    to_NET.seg.syn_flag := FALSE;
    to_NET.seg.data_offset := OPTIONLESS_HEADER;
    to_NET.seg.wndw := sv.rcv_wndw;

  --Note that this section assumes that this segment's size
  --is less than sv.send_max_seg.

  --The end of pushed data cannot be packaged with
  --subsequent non-pushed data.

  --Prepare and transmit data.

  ds_copy_from_send( sv.send_una, retrans_amount );

  --If pushed data within or following data in this segment,
  --set the PUSH flag to inform remote TCP.
  if (sv.send_una < sv.send_push)

```

MIL-STD-1778
12 August 1983

```

then to_NET.seg.push_flag := TRUE
else to_NET.seg.push_flag := FALSE;
--If urgent data lies within or follows data in this segment,
--record urgent data position in header.
if (sv.send_urg > start_pt)
then to_NET.seg.urg_flag := TRUE;
   to_NET.seg.urgptr := sv.send_urg - start_pt;
else to_NET.seg.urg_flag := FALSE;
--If this segment contains that last octet of data from
--the ULP, set the FIN to inform the remote TCP.
if ((sv.send_finflag = TRUE) and
    (sv.send_free = start_pt + retrans_amount))
then to_NET.seg.fin_flag := TRUE
else to_NET.seg.fin_flag := FALSE;

format_net_params( retrans_amount );
compute_checksum;
TRANSFER to_NET to the network protocol entity;
end; --of preparation and retransmission of "unpushed data.
end;
```

9.4.6.3.37 Retransmit policy. As one of the policy procedures, retransmit policy discusses the alternative strategies for retransmissions. It returns to the calling action procedure the number of octets to be retransmitted. A TCP implementation may employ one of several retransmission strategies.

- a. First only retransmission - Maintain one retransmission timer for the entire queue. When the retransmission timer expires, send the segment at the front of the retransmission queue. Initialize the timer.
- b. Batch retransmission - Maintain one retransmission timer for the entire queue. When the retransmission timer expires, send all the segments on the retransmission queue. Initialize the timer.
- c. Individual retransmission - Maintain one timer for each segment on the retransmission queue. As the timers expire, retransmit the segments individually and reset their timers.

9.4.6.3.37.1 Retransmission strategy. The first only retransmission strategy is efficient in terms of traffic generated because only lost segments are retransmitted; but the strategy can cause long delays. The batch retransmission creates more traffic but decreases the likelihood of long delays. However, the actual effectiveness of either scheme depends in part on the acceptance policy of the receiving TCP. For example, suppose a sending TCP sends three segments, all within the send window, to a receiving TCP. The first segment is lost by the network. A receiving TCP using the "in-order" acceptance strategy discards the second and third segments. A receiving TCP

MIL-STD-1778
12 August 1983

using the "in-window" strategy accepts the second and third segments, but does not acknowledge or deliver any data until the lost segment arrives. Batch retransmission fits better with the in-order acceptance strategy because the receiving TCP has discarded all segments. The sooner all three segments are retransmitted, the better. First-only retransmission fits better with the in-window acceptance policy because only the needed retransmission occurs because the receiving TCP has kept the segments within its receive window and awaits only the lost segment. The sending TCP may also choose to repackage segments for retransmission.

9.4.6.3.38 Save fin. The save_fin action procedure records the presence of a FIN flag in an incoming segment received before a connection is ESTABLISHED. The FIN is processed only in the ESTABLISHED state. The data effects of the procedure are:

- a. Data examined: sv.recv_next
 - b. Data modified: sv.recv_fin sv.recv_push
- Record FIN is recv variable.
sv.recv_finflag := TRUE;
sv.recv_push := sv.recv_next; --The PUSH function is assumed.

9.4.6.3.39 Save send data. The save_send_data action procedure saves the data provided by the local ULP in a "Send" or an "Active Open with Data" service request issued before the connection is ESTABLISHED. The data effects of the procedure are:

- a. Data examined only:

<u>from_ULP.data</u>	<u>from_ULP.length</u>
<u>from_ULP.push_flag</u>	<u>from_ULP.urgent_flag</u>
 - b. Data modified:

<u>sv.send_free</u>	<u>sv.send_urg</u>
<u>sv.send_push</u>	
- begin
 --Take the data and add it to the send queue.
dm_add_to_send(sv.send_free, from_ULP.length);
sv.send_free := sv.send_free + from_ULP.length;

 --Set the urgent and push information as needed.
 if (from_ULP.push_flag = TRUE)
 then sv.send_push := sv.send_free;

 if (from_ULP.urg_flag = TRUE)
 then sv.send_urg := sv.send_free;
end;

MIL-STD-1778
12 August 1983

9.4.6.3.40 Send ack. The send_ack procedure formats and sends an empty segment with the ACK value indicated by parameter. The data effects of this procedure are:

```

a. Data examined:
    sv.send_next      sv.source_port
    sv.recv_next      sv.destination_port
    sv.actual_prec     sv.sec

b. Data modified: all to_NET.segment fields

begin
--The ACK field of the segment is set to the parameter value.
  to_NET.segment.ack_flag := TRUE;
  to_NET.segment.ack_num  := parameter;

--Fill in the rest of the segment and network parameters.
  to_NET.segment.seq_num  := sv.send_next;
  to_NET.segment.reset_flag := FALSE;
  to_NET.segment.syn_flag := FALSE;
  to_NET.segment.push_flag := FALSE;
  to_NET.segment.fin_flag := FALSE;
  to_NET.segment.data_offset := OPTIONLESS_HEADER;
  to_NET.segment.window    := sv.recv_wndw;

--Add security and precedence information to header.
--Add in urgent information if needed.
  if (sv.send_urg > to_NET.segment.seq_num)
  then --record urgent data position in header
    to_NET.segment.urg_flag := TRUE;
    to_NET.segment.urgptr   := sv.send_urg - to_NET.segment.seq_num;
  else to_NET.segment.urg_flag := FALSE;

  format_net_params( 0 );
  compute_checksum;
  TRANSFER to_NET to the network protocol entity;

--Adjust implementation dependent ACK parameters such as
--ACK timer, or state_vector element for the last ACK'd octet.
end;
```

9.4.6.3.41 Send fin. The send_fin action procedure records a close request and, if no data is waiting to be transmitted, formats and sends an empty segment with the FIN flag set. If data is waiting and the window permits, the FIN is sent along with the data. The data effects of this procedure are:

```

a. Data examined: sv.send_next

b. Data modified: sv.send_finflag  sv.send_push

--Record the CLOSE service request. The CLOSE implies a PUSH.
  sv.send_finflag := TRUE;
  sv.send_push := sv.send_next;
```

MIL-STD-1778
12 August 1983

--The FIN is sent along with any waiting data.
send_new_data;

9.4.6.3.42 Send new data. The send_new_data action procedure examines the send window, the amount of pushed data, and segment size restrictions to determine if any waiting data can be sent to the remote TCP. The data effects of this procedure are:

- a. Data examined:

sv.send_max_seg	sv.recv_next
sv.source_port	sv.recv_wndv
sv.destination_port	sv.send_finflag
- b. Data modified:

sv.send_next	sv.send_push
sv.send_free	sv.send_urg
sv.send_wndv	all fields of to_NET
- c. Local variables: send_amount

```
begin
  --The amount of data to be sent is determined by the
  --send window, the amount of data waiting, the amount of
  --pushed data, and segment size restrictions.

  if ((sv.send_wndv /= 0) and (sv.send_next /= sv.send_free))
  then begin
    --Data can be sent, but how much?
    --Check for pushed data, which must be sent as soon
    --as the window allows.
    begin
      if (sv.send_push > sv.send_next)
      then --Pushed data awaits transmission

        if (sv.send_push < sv.send_una + sv.send_wndv)
        then --all pushed data can be sent
          send_amount := sv.send_push - sv.send_next;
          to_ULP.seg.push_flag := TRUE;
        else --send all pushed data allowed by send window
          send_amount := sv.send_una + sv.send_wndv - sv.send_next;
          to_NET.seg.push_flag := FALSE;
        else --No pushed data waiting. Refer to send policy
          --to determine amount (if any) to be sent.
          send_amount := send_policy();
          to_NET.seg.push_flag := FALSE;

      --How much data to send has been determined. Now
      --format and transmit the segment.
      if (send_amount > 0)
      then begin
```

MIL-STD-1778
12 August 1983

```

to_NET.seq.seq_num      := sv.send_next;
to_NET.seq.ack_num      := sv.recv_next;
to_NET.seq.ack_flag     := TRUE;
to_NET.seq.syn_flag     := FALSE;
to_NET.seq.rst_flag     := FALSE;
to_NET.seq.data_offset  := OPTIONLESS_HEADER;
to_NET.seq.window       := sv.recv_wndw;
--Add security and precedence to header.
--The ULP may have already CLOSED. If so, and this
--data includes the last octet, set the FIN.
if ((sv.send_finflag = TRUE) and
    (sv.send_free = to_NET.seq.seq_num + send_amount))
then to_NET.seq.fin_flag := TRUE
else to_NET.seq.fin_flag := FALSE;

if (sv.send_urg > to_NET.seq.seq_num)
then --record urgent data position in header
    to_NET.seq.urg_flag := TRUE;
    to_NET.seq.urgptr := sv.send_urg - to_NET.seq.seq_num;
else to_NET.seq.urg_flag := FALSE;

da_copy_from_send( sv.send_next, send_amount );
sv.send_next := sv.send_next + send_amount;

format_net_params( send_amount );
compute_checksum;
TRANSFER to_NET to the network protocol entity;

--Depending on the retransmission policy chosen for
--an implementation, a retransmission timer
--may now need to be set for the newly sent data.
--implementation dependent action

end; --of preparation and transmission of data.
end;
end;
```

9.4.6.3.43 Send policy. Barring pushed data and zero receive windows, the TCP entity is left to segment and transfer data at its convenience. The number of octets that should be sent beginning at sv.send_next is returned to the calling procedure. The definition of "convenience" should be influenced by design goals. If the primary goal is low overhead in terms of segment generation, then data should be accumulated until a maximum segment's worth (defined by the remote TCP) is ready. However, if quick response is the main goal, the TCP entity should segment and transmit data at regular intervals to minimize delay. Another aspect of the send policy is related to window management. Discussed in the paragraph 9.2.3, the handling of small send windows may alter sending behavior. The TCP entity may choose to avoid sending into small windows (where small is defined as a percentage of segment size or storage capacity) to achieve better throughput.

MIL-STL-1718
17 August 1983

9.4.6.3.44 Set fin. The set fin action procedure records the presence of a FIN in an incoming segment. The ULP is informed of the remote ULP's CLOSE after all data from the remote ULP is delivered. The data effects of this procedure are:

```

a. Data examined:  sv.recv_save      sv.recv_next
b. Data modified:  sv.recv_finflag

begin
  --Record the FIN's presence for use in the ESTABLISHED state.
  sv.recv_finflag := TRUE;
  sv.recv_push := sv.recv_next;

  --If no data is waiting to be delivered, a CLOSING
  --service response is issued to inform the local ULP of the
  --remote ULP's CLOSE request.

  if (sv.recv_save = sv.recv_next)
    and (no data is awaiting re-ordering)
  then
    to_ULP.service_response := CLOSING;
    to_ULP.lcn := sv.lcn;
    TRANSFER to_ULP to the ULP named by sv.source_port.
end;
```

9.4.6.3.45 Start time wait. The start time wait action procedure cancels all other timers and sets the final "TIME_WAIT" timer which allows time for the final FIN acknowledgment to reach the remote TCP before clearing the state vector of this connection. The data effects of this procedure are:

```

a. Data examined: - none -
b. Data modified: - none -

begin
  --Issue timer cancellation requests to the execution environment
  --corresponding to all current timers.
  cancel_timer( ULP_TIMEOUT );
  cancel_timer( RETRANSMIT );

  --Depending on implementation strategies, ACK timers and
  --zero window timers may also exist.

  --Start up the time_wait timer for the appropriate duration--currently
  --suggested to be 2 minutes.

  start_timer( TIME_WAIT, TIME_WAIT_INTERVAL );
end;
```

MIL-STD-1778
12 August 1983

9.4.6.3.46 Update. The update routine takes a new ACK from the incoming segment to update the send and receive variables. The data effects of this procedure are:

- a. Data examined:

from_NET.seg.ack_num	from_NET.seg.window
from_NET.seg.seq_num	
- b. Data modified:

sv.send_una	sv.send_wndw
sv.send_lastup1	sv.send_lastup2

begin

--Take only new ACKs, i.e. those greater than sv.send_una.

```
if from_NET.seg.ack_num > sv.send_una
then begin  --update the retransmission queue
  dm_remove_from_send(sv.send_una, (from_NET.seg.ack_num -
  sv.send_una)); sv.send_una := from_NET.seg.ack_num;
```

--Depending on retransmission strategy, the retransmission
--timer may need resetting because of the new ACK.
--implementation dependent action

--The retransmission timeout interval may need adjustment
--to adapt to the round-trip time of the data just ACK-ed.
--implementation dependent action

--The ULP timeout timer may need resetting due to the
--the successful delivery of the newly ACK-ed data.
--implementation dependent action

end;

--A new window is provided if either the sequence number of this
--segment is newer than the one last used to update the window, or
--(for 1-way data transfer) the sequence number is the same but
--the ACK is greater.

```
if ((sv.send_lastup1 < from_NET.seg.seq_num) or
    (sv.send_lastup2 < from_NET.seg.ack_num))
then begin
  sv.send_wndw := (from_NET.seg.ack_num + from_NET.seg.window)
                 - sv.send_una;
  sv.send_lastup1 := from_NET.seg.seq_num;
  sv.send_lastup2 := from_NET.seg.ack_num;
```

--Because a new send window has arrived, try to send data.
send_new_data;

end;

end;

MIL-STD-1778
12 August 1983

10. EXECUTION ENVIRONMENT REQUIREMENTS

10.1 Introduction. Throughout this document, the environmental model portrays each protocol entity acting as an independent process. Within this model, the execution environment must provide two facilities: inter-process communication and timing.

10.2 Inter-process communication. The execution environment must provide an inter-process communication facility to enable independent processes to pass units of information, called messages. For TCP's purposes, the IPC facility is required to preserve the order of messages. TCP uses the IPC facility to exchange interface parameters and data with upper layer protocols across its upper interface and the network protocol across the lower interface. Sections 6 and 7 specify these interfaces. In the service and entity specifications, this service is accessed through a the following primitive: TRANSFER - passes a message to a named target process.

10.3 Timing. The execution environment must provide a timing facility that maintains 32-bit clock (possibly fictitious) with units no coarser than 1 second. A process must be able to set a timer for a specific time period and be informed by the execution environment when the time period has elapsed. A process must also be able to cancel a previously set timer. Several TCP mechanisms use the timing facility. The positive acknowledgment with retransmission mechanism uses timers to ensure that if data or acknowledgments are lost, they are re-sent. The ULN timeout mechanism uses the timing facility to clock the delay between data transmission and acknowledgment. The time-wait mechanism uses a timer to allow enough time for a final FIN acknowledgement to arrive at the remote TCP entity before connection termination. Other uses for a timing facility are implementation dependent. In the upper service and entity specification, the timing services are accessed with the following primitives:

- a. SET_TIMER (timer_name, time_interval) - allows a given interval of time and an identifier to be specified. After the specified interval elapses, and timeout indication and the identifier is returned to the issuing process.
- b. CANCEL_TIMER (timer_name) - allows the timeout associated with the identifier to be terminated.
- c. CURRENT_TIME - returns the current time.

Custodians:

Army - CR
Navy - OM
Air Force - 90

Preparing Activity:

DCA-DC
(Project JPSC-0178-02)

Review Activities:

Army - SC, CR, AD
Navy - AS, YD, MC, OM, ND, NC, EC, SA
Air Force - 1, 11, 13, 17, 90, 99

Other Interest:

NSA-NS
TRI-TAC-TT

MIL-STD-1778
12 August 1983

APPENDIX A. RETRANSMISSION STRATEGY EFFECTIVENESS

As noted in the entity overview, Section 9.2, a TCP implementation may employ one of several retransmission strategies:

- a. First-only retransmission - A TCP maintains one retransmission timer for the queue, retransmitting the front segment (or segment's worth of data) when the timer expires.
- b. Batch retransmission - A TCP maintains one retransmission timer for the queue, retransmitting all segments on the queue when the timer expires.
- c. Individual retransmission - A TCP maintains one timer per segment on the queue, retransmitting each segment when its individual timer expires.

The first-only retransmission strategy is efficient in terms of traffic generated because only lost segments are retransmitted; but the strategy can cause long delays. The batch retransmission creates more traffic but decreases the likelihood of long delays. The individual retransmission strategy is a compromise between delay and traffic but requires much more processing time from the TCP entity. However, the actual effectiveness of each scheme depends in part on the acceptance policy (paragraph 9.2.4) of the receiving TCP.

For example, suppose a sending TCP sends three segments, all within the send window, to a receiving TCP. The first segment is lost by the network. A receiving TCP using the "in-order" acceptance strategy discards the second and third segments. A receiving TCP using the "in-window" strategy accepts the second and third segments, but does not acknowledge or deliver any data until the intervening segment arrives.

Batch retransmission performs better with the in-order acceptance strategy because the receiving TCP has discarded all segments. All three segments must be retransmitted--the sooner the better. First-only retransmission performs better with the in-window acceptance policy because only the necessary retransmissions occur since the receiving TCP has kept the segments within its receive window and awaits only the lost segment.

Unfortunately, a sending TCP cannot know what acceptance policy is being used by the receiving TCP. Instead, the retransmission strategy must be chosen according to implementation dependent and configuration dependent design goals.

MIL-STD-1778
12 August 1983

APPENDIX B. DYNAMIC RETRANSMISSION TIMER COMPUTATION

Because of the variability of the networks that compose the internetwork system and the wide range of uses of TCP connections, the retransmission timeout should be dynamically determined. One procedure for determining a retransmission time out is given here as an illustration.

Measure the elapsed time between sending a data octet with a particular sequence number and receiving an acknowledgment that covers that sequence number (Segments sent do not have to match segments received). This measured elapsed time is the Round Trip Time. Next, compute a Smoothed Round Trip Time (SRTT) as:

$$SRTT = (\text{ALPHA} * SRTT) + ((1-\text{ALPHA}) * RTT)$$

and based on this, compute the retransmission timeout (RTO) as:

$$RTO = \text{minimum}(\text{UBOUND}, \text{maximum}(\text{LBOUND}, (\text{BETA} * SRTT)))$$

where:

UNBOUND = an upper bound on the timeout (e.g., 1 minute)

LBOUND = a lower bound on the timeout (e.g., 1 second)

ALPHA = a smoothing factor (e.g., .8 to .9)

BETA = a delay variance factor (e.g., 1.3 to 2.0)

MIL-STD-1778
12 August 1983

APPENDIX C. ALTERNATIVES IN SERVICE INTERFACE PRIMITIVES

The service primitives offered to the upper level protocol are specified in paragraph 6.2. The service request primitives are:

- Unspecified Passive Open,
- Fully Specified Passive Open,
- Active Open,
- Active Open with Data,
- Send,
- Allocate,
- Status,
- Close, and
- Abort

These primitives support the minimal services required of a TCP. However, combinations or modifications may offer additional services that are tailored to the requirements of a particular set of upper level protocols. Several examples are provided below.

If the protocol supporting TCP is the Internet Protocol and a TCP implementation wishes to export IP's option services (including source routing, record routing, stream identification and timestamps), an additional "options" parameter would be required in all Open and Send service requests.

An upper level protocol may need a reliable transaction service. That is, a ULP may wish to open a connection, send a single message, and then close the connection. To access this service, the specified service interface requires the ULP to issue at least two service primitives, an Open with Data and a Close, to exercise this service. A TCP may be designed with a service primitive that combined the Open and Close to form a new primitive, called perhaps Transaction, which would include all the Open parameters, the data to be transmitted, and the signal to close the connection after data delivery.

The upper layer service definition (paragraph 6.3) does not allow a Passive Open request to be followed by an Active Open request. Instead, the ULP must first issue a Close or Abort request to cancel the Passive Open request, then issue an Active Open request. A TCP may be designed to allow "conversion" of open requests from passive to active. In this case, a ULP could issue a Full Passive Open request followed by an Active Open or a Send request to actively initiate a connection. Thus, the local entity service diagram (appearing in paragraph 6.4) changes to include a transition from the PASSIVE to the ACTIVE state as shown in Figure 15.

MIL-STD-1778
12 August 1983

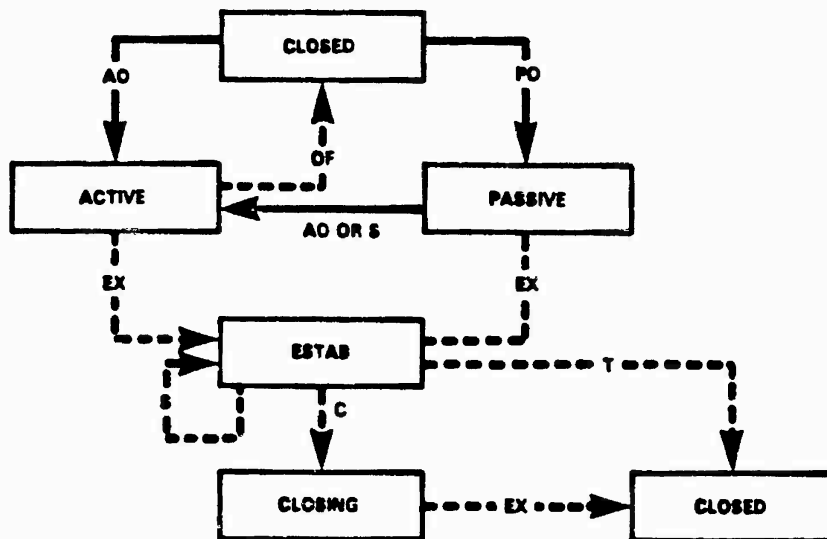


FIGURE 16. TCP local service state machine summary.

U.S. GOVERNMENT PRINTING OFFICE: 1983-603-034-0027

MIL-STD-1780
10 May 1984

MILITARY STANDARD

FILE TRANSFER PROTOCOL



NO DELIVERABLE DATA
REQUIRED BY THIS DOCUMENT

IPSC/SLMC/TCTS

MIL-STD-1780
10 May 1984

DEPARTMENT OF DEFENSE
WASHINGTON, D.C. 20301

File Transfer Protocol

MIL-STD-1780

1. This Military Standard is approved for use by all Departments and Agencies of the Department of Defense.

2. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to: Defense Communications Engineering Center, ATTN: R130, 1850 Wiehle Avenue, Reston, Virginia 22090-5500, by using the self-addressed Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document, or by letter.

MIL-STD-1780
10 May 1984

FOREWORD

This document specifies the File Transfer Protocol (FTP) which supports the transfer of file data throughout a heterogeneous host computer network. This draft standard defines the FTP's role and purpose, defines the services provided to users and specifies the mechanisms needed to support these services.

MIL-STD-1780
10 May 1984

CONTENTS

		<u>Page</u>
Paragraph 1.	SCOPE - - - - -	1
1.1	Purpose- - - - -	1
1.2	Organization - - - - -	1
1.3	Application- - - - -	1
1.4	Objectives - - - - -	1
2.	REFERENCED DOCUMENTS- - - - -	2
2.1	Issues of documents- - - - -	2
2.2	Other publications - - - - -	2
3.	DEFINITIONS - - - - -	3
3.1	Definition of terms- - - - -	3
4.	GENERAL REQUIREMENTS- - - - -	6
4.1	General- - - - -	6
4.2	The FTP model- - - - -	6
4.2.1	TELNET initiation- - - - -	7
4.3	FTP data connection- - - - -	7
4.3.1	TELNET connections - - - - -	7
4.4	Data transfer functions- - - - -	8
5.	DATA REPRESENTATION AND STORAGE - - - - -	9
5.1	Data representation differences- - - - -	9
5.1.1	Data representation in binary- - - - -	9
5.2	Data representation types- - - - -	9
5.2.1	ASCII format - - - - -	9
5.2.2	EBCDIC format- - - - -	10
5.3	Reasons for file transfer- - - - -	10
5.3.1	File transfer parameters - - - - -	10
5.3.1.1	Non-print- - - - -	10
5.3.1.2	TELNET format controls - - - - -	10
5.3.1.3	Carriage control (ASA) - - - - -	10
5.3.1.4	Image- - - - -	11
5.3.1.5	Local byte size- - - - -	11
5.3.1.5.1	Local byte example 1 - - - - -	11
5.3.1.5.2	Local byte example 2 - - - - -	11
5.3.1.6	File parameter caution - - - - -	11
5.3.2	File structure - - - - -	12
5.3.2.1	File- vs. record-oriented- - - - -	12
5.3.2.2	Page structure - - - - -	13
5.4	Establishing data connections- - - - -	14
5.4.1	Passive data transfer process- - - - -	14
5.4.2	Alternate data - - - - -	14
5.5	Transmission modes - - - - -	15
5.5.1	Transmission modes defined - - - - -	15

MIL-STD-1780
10 May 1984

CONTENTS - Continued

		<u>Page</u>
Paragraph	5.5.1.1 Stream - - - - -	15
	5.5.1.2 Block- - - - -	16
	5.5.1.2.1 Descriptor codes - - - - -	16
	5.5.1.3 Compressed - - - - -	17
	5.6 Error recovery and restart - - - - -	18
	5.7 File transfer functions- - - - -	19
	5.7.1 FTP commands - - - - -	19
	5.7.1.1 Access control commands- - - - -	19
	5.7.1.1.1 User name (USER) - - - - -	19
	5.7.1.1.2 Password (PASS)- - - - -	19
	5.7.1.1.3 Account (ACCT) - - - - -	20
	5.7.1.1.4 Reinitialize (REIN)- - - - -	20
	5.7.1.1.5 Logout (QUIT)- - - - -	20
	5.7.2 Transfer parameter commands- - - - -	20
	5.7.2.1 Data port (PORT) - - - - -	21
	5.7.2.2 Passive (PASV) - - - - -	21
	5.7.2.3 Representation type (TYPE) - - - - -	21
	5.7.2.4 File structure (STRU)- - - - -	21
	5.7.2.5 Transfer mode (MODE) - - - - -	22
	5.7.3 FTP service commands - - - - -	22
	5.7.3.1 Retrieve (RETR)- - - - -	22
	5.7.3.2 Store (STOR) - - - - -	22
	5.7.3.3 Append (with create) (APPE)- - - - -	22
	5.7.3.4 Allocate (ALLO)- - - - -	22
	5.7.3.5 Restart (REST) - - - - -	23
	5.7.3.6 Rename from (RNFR) - - - - -	23
	5.7.3.7 Rename to (RNTD) - - - - -	23
	5.7.3.8 Abort (ABOR) - - - - -	23
	5.7.3.8.1 FTP service command completed- - - - -	23
	5.7.3.8.2 FTP service command in progress- - - - -	23
	5.7.3.10 Delete (DELE)- - - - -	23
	5.7.3.11 Change working directory (CWD) - - - - -	24
	5.7.3.12 List (LIST)- - - - -	24
	5.7.3.13 Name-list (NLST) - - - - -	24
	5.7.3.14 Site parameters (SITE) - - - - -	24
	5.7.3.15 Status (STAT)- - - - -	24
	5.7.3.16 Help (HELP)- - - - -	24
	5.7.3.17 Noop (NOOP)- - - - -	25
	5.7.4 TELNET language- - - - -	25
	5.8 FTP replies- - - - -	25
	5.8.1 FTP reply defined- - - - -	26
	5.8.2 Three-digit reply defined- - - - -	27
	5.8.2.1 First digit values - - - - -	27
	5.8.2.1.1 Positive preliminary reply (1yz) - - - - -	27

MIL-STD-1780
10 May 1984

CONTENTS - Continued

	<u>Page</u>
Paragraph 5.8.2.1.2 Positive completion reply (2yz)- - - - -	27
5.8.2.1.3 Positive intermediate reply (3yz)- - - - -	27
5.8.2.1.4 Transient negative completion reply (4yz)- -	27
5.8.2.1.5 Permanent negative completion reply (5yz)- -	28
5.8.2.2 Second digit values- - - - -	28
5.8.2.2.1 Syntax (x0z)- - - - -	28
5.8.2.2.2 Information (x1z)- - - - -	28
5.8.2.2.3 Connections (x2z)- - - - -	28
5.8.2.2.4 Authentication and accounting (x3z)- - - -	28
5.8.2.2.5 Unspecified (x4z)- - - - -	28
5.8.2.2.6 File system (x5z)- - - - -	28
5.8.2.3 Third digit values - - - - -	28
5.8.3 Reply codes by function groups - - - - -	29
5.8.4 Numeric order list of reply codes- - - - -	31
5.9 Declarative specifications - - - - -	33
5.9.1 Minimum implementation - - - - -	33
5.9.2 Connections- - - - -	34
5.9.2.1 Command-reply sequence - - - - -	34
5.9.3 Commands - - - - -	35
5.9.3.1 FTP command list - - - - -	35
5.9.3.2 FTP command syntax - - - - -	36
5.10 Sequencing of commands and replies - - - -	36
5.10.1 Command-reply sequences- - - - -	37
5.11 State diagrams - - - - -	40
5.11.1 largest group of FTP commands- - - - -	40
5.11.2 Other large group of FTP commands- - - - -	40
5.11.3 Rename sequence command- - - - -	41
5.11.4 Restart command- - - - -	42
5.11.5 login sequence - - - - -	43
5.11.6 Command and reply interchange- - - - -	44
6. TYPICAL FTP SCENARIO- - - - -	45
6.1 Scenario - - - - -	45
6.2 Connection establishment - - - - -	45

MIL-STD-1780

10 May 1984

FIGURES

		<u>Page</u>
Figure 1	Example FTP configuration in a host	
	protocol hierarchy- - - - -	6
2	Model for FTP use- - - - -	6
3	Server-server interaction- - - - -	7
4	Block header - - - - -	16
5	Transmission of a six-character marker - - - -	17
6	Byte string- - - - -	17
7	Replicated byte- - - - -	18
8	Filler string- - - - -	18
9	Largest group of FTP commands- - - - -	40
10	Other group of FTP commands- - - - -	40
11	Rename sequence- - - - -	41
12	Restart command- - - - -	42
13	Login sequence - - - - -	43
14	Command and reply interchange- - - - -	44

MIL-STD-1780
10 May 1984

1. SCOPE

1.1 Purpose. This standard establishes criteria for the File Transfer Protocol (FTP) used for transferring files between computer systems.

1.2 Organization. This standard introduces the File Transfer Protocol's role and purpose, defines the services provided to users, and specifies the mechanisms needed to support those services.

1.3 Application. The File Transfer Protocol is approved for implementation in hosts of all DoD packet switching networks which connect or have the potential for utilizing connectivity across network and subnetwork boundaries and which require a file transfer service. The term network as used herein includes Local Area Networks.

1.4 Objectives. The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among Hosts, and 4) to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs. The attempt in this specification is to satisfy the diverse needs of computer users, with a simple and easily implemented protocol design. This standard assumes knowledge of Transmission Control Protocol, MIL-STD-1778, and TELNET Protocol, MIL-STD-1782.

MIL-STD-1780
10 May 1984

2. REFERENCED DOCUMENTS

2.1 Issues of documents. The following documents of the issue in effect on date of invitation for bids or request for proposal, form a part of this standard to the extent specified herein. (The provisions of this paragraph are under consideration.)

Standards:

Federal

FED-STD-1037 Glossary of Telecommunications Terms

Military

MIL-STD-1778 Transmission Control Protocol
MIL-STD-1782 TELNET Protocol

2.2 Other publications. The following documents form a part of this standard to the extent specified herein. Unless otherwise indicated, the issue in effect on date of invitation for bids or request for proposal shall apply. (The provisions of this paragraph are under consideration.)

MIL-STD-1780
10 May 1984

- s. Server-PI. The protocol interpreter "listens" on Port L for a connection from a user-PI and establishes a TELNET communication connection. It receives standard FTP commands from the user-PI, sends replies, and governs the server-DTP.
- t. TELNET connections. The full-duplex communication path between a user-PI and a server-PI, operating according to the TELNET Protocol.
- u. Type. The data representation type used for data transfer and storage. Type implies certain transformations between the time of data storage and data transfer. The representation types defined in FTP are described in the Section on Establishing Data Connections.
- v. User-DTP. The data transfer process "listens" on the data port for a connection from a server-FTP process. If two servers are transferring data between them, the user-DTP is inactive.
- w. User-FTP process. A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes. The user interface allows a local language to be used in the command-reply dialogue with the user.
- x. User-PI. The protocol interpreter initiates the command connection from its port U to the server-FTP process, initiates FTP commands, and governs the user-DTP if that process is part of the file transfer.

MIL-STD-1780
10 May 1984

4. GENERAL REQUIREMENTS

4.1 General. In specifying a File Transfer Protocol it is desirable not to assume a set system configuration. As a practical matter, the distributions of the file transfer protocol will vary with specific hardware configurations. Although appearing to focus on FTP implementations, this standard can apply to any configuration given appropriate protocols to bridge hardware boundaries. An example of where FTP is configured in a host protocol hierarchy can be seen in Figure 1.

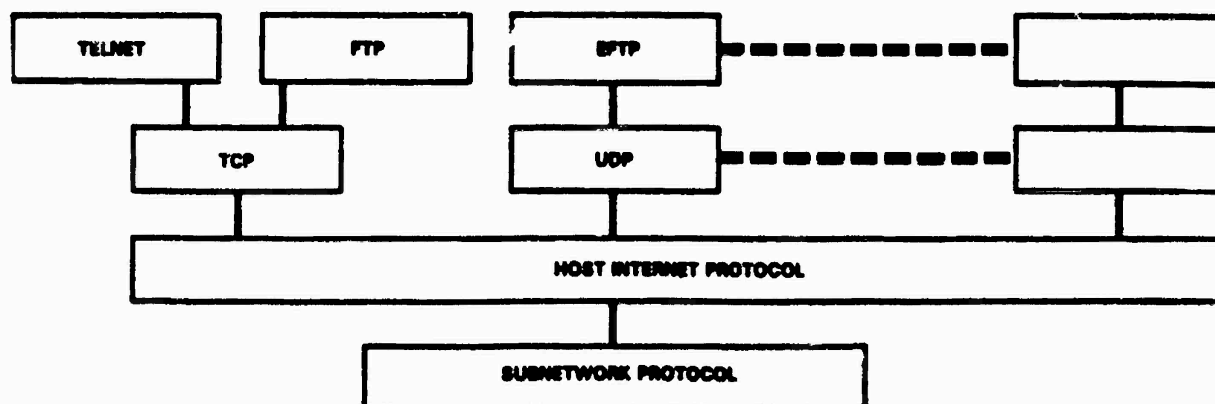
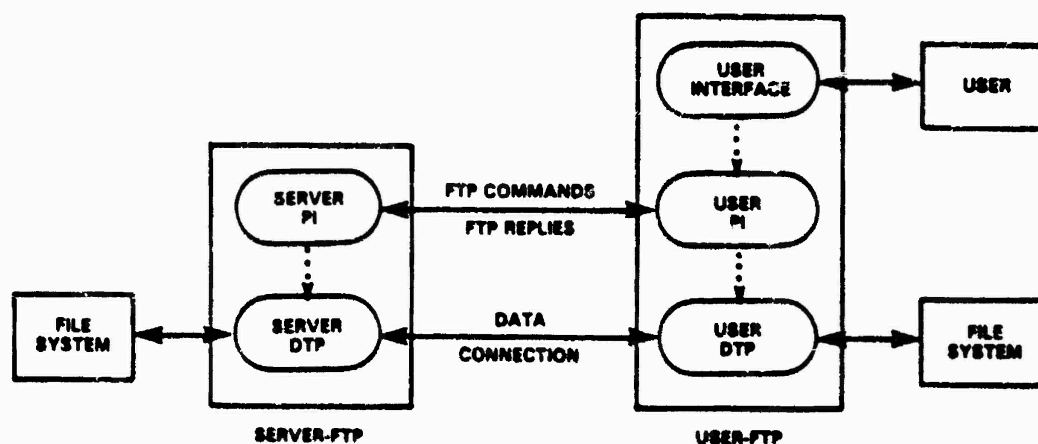


FIGURE 1. Example FTP configuration in a host protocol hierarchy.

4.2 The FTP model. The following model (see Figure 2) may be diagrammed for an FTP service.



NOTES: 1. The data connection may be used in either direction.
2. The data connection need not exist all of the time.

FIGURE 2. Model for FTP use.

MIL-STD-1780
10 May 1984

3. DEFINITIONS

3.1 Definition of terms. The definition of terms used in this standard shall comply with FED-STD-1037. Terms and definitions unique to MIL-STD-1780 are contained herein.

- a. Byte size. There are two byte sizes of interest in FTP: the logical byte size of the file, and the transfer byte size used for the transmission of the data. The transfer byte size is always 8 bits. The transfer byte size is not necessarily the byte size in which data is to be stored in a system, nor the logical byte size for interpretation of the structure of the data.
- b. Command Connection. A TELNET connection in which FTP commands and replies are exchanged between the user and server FTP processes, respectively. Although the degree to which the TELNET protocol is utilized is not specified, both user and server processes must have the capability to participate in TELNET negotiations. (The default port for the command connection is defined as Port L.)
- c. Data connection. A simplex connection over which data is transferred, in a specified mode and type. The data transferred may be a part of a file, an entire file or a number of files. The path may be between a server-DTP and a user-DTP, or between two server-DTPs.
- d. Data port. The passive data transfer process "listens" on the data port for a connection from the active transfer process in order to open the data connection.
- e. EOF. The end-of-file condition that defines the end of a file being transferred.
- f. EOR. The end-of-record condition that defines the end of a record being transferred.
- g. Error recovery. A procedure that allows a user to recover from certain errors such as failure of either Host system or transfer process. In FTP, error recovery may involve restarting a file transfer at a given checkpoint.
- h. FTP commands. A set of commands that comprise the control information flowing from the user-FTP to the server-FTP process.
- i. File. An ordered set of computer data (including programs), of arbitrary length, uniquely identified by a pathname.

MIL-STD-1780
10 May 1984

- j. Mode. The mode in which data is to be transferred via the data connection. The mode defines the data format during transfer including EOR and EOF. The transfer modes defined in FTP are described in the Section on Transmission Modes.
- k. NVT. The Network Virtual Terminal as defined in the TELNET Protocol (paragraph 4.2.1, MIL-STD-1782.)
- l. NVFS. The Network Virtual File System. A concept which defines a standard network file system with standard commands and pathname conventions. FTP only partially implements the NVFS concept at this time.
- m. Page. A file may be structured as a set of independent parts called pages. FTP supports the transmission of discontinuous files as independent indexed pages.
- n. Pathname. Pathname is defined to be the character string which must be input to a file system by a user in order to identify a file. Pathname normally contains device and/or directory names, and file name specification. FTP does not yet specify a standard pathname convention. Each user must follow the file naming conventions of the file systems involved in the transfer.
- o. Record. A sequential file may be structured as a number of contiguous parts called records. Record structures are supported by FTP but a file need not have record structure.
- p. Reply. A reply is an acknowledgment (positive or negative) sent from server to user via the command connections in response to FTP commands. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.
- q. Server-DTP. The data transfer process, in its normal "active" state, establishes the data connection with the "listening" data port, sets up parameters for transfer and storage, and transfers data on command from its PI. The DTP can be placed in a "passive" state to listen for, rather than initiate, a connection on the data port.
- r. Server-FTP process. A process or set of processes which perform the function of file transfer in cooperation with a user-FTP process and, possibly, another server. The functions consist of a protocol interpreter (PI) and a data transfer process (DTP).

MIL-STD-1780
10 May 1984

4.2.1 Command initiation. In Figure 2, the user-protocol interpreter initiates the command connection. At the initiation of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the command connection. (The user may establish a direct command connection to the server-FTP, from a TAC terminal for example, and generate standard FTP commands himself, bypassing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the command connection in response to the commands.

4.3 FTP data connections. The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The user-DTP or its designate should "listen" on the specified data port, and the server initiate the data connection and data transfer in accordance with the specified parameters. It should be noted that the data port need not be in the same Host that initiates the FTP commands via the command connection, but the user or the user-FTP process must ensure a "listen" on the specified data port. It should also be noted that the data connection may be used for simultaneous sending and receiving. In another situation, a user might wish to transfer files between two Hosts, neither of which is his local Host. He sets up command connections to the two servers and then arranges for a data connection between them. In this manner control information is passed to the user-PI but data is transferred between the server data transfer processes. Figure 3 is a model of this server-server interaction.

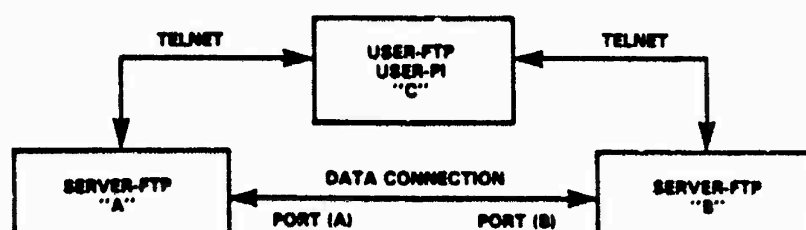


FIGURE 3. Server-server interaction.

4.3.1 Command connections. The protocol requires that the command connections be open while data transfer is in progress. It is the responsibility of the user to request the closing of the command connections when finished using the FTP service, while it is the server who takes the action. The server may abort data transfer if the command connections are closed without command.

MIL-STD-1780
10 May 1984

4.4 Data transfer functions. Files are transferred only via the data connection. The command connection is used for the transfer of commands, which describe the functions to be performed, and the replies to these commands (see the Section on FTP Replies). Several commands are concerned with the transfer of data between Hosts. These data transfer commands include the MODE command which specify how the bits of the data are to be transmitted, and the STRUcture and TYPE commands, which are used to define the way in which the data are to be represented. The transmission and representation are basically independent but "Stream" transmission mode is dependent on the file structure attribute and if "Compressed" transmission mode is used the nature of the filler byte depends on the representation type.

MIL-STD-1780
10 May 1984

5. DATA REPRESENTATION AND STORAGE

5.1 Data representation differences. Data is transferred from a storage device in the sending Host to a storage device in the receiving Host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. For example, NVT-ASCII has different data storage representations in different systems. A computer may store NVT-ASCII as five 7-bit ASCII characters, left-justified in a 36-bit word. Another system may store NVT-ASCII as 8-bit EBCDIC codes. It may be desirable to convert characters into the standard NVT-ASCII representation when transmitting text between dissimilar systems. The sending and receiving sites would have to perform the necessary transformations between the standard representation and their internal representations.

5.1.1 Data representation in binary. A different problem in representation arises when transmitting binary data (not character codes) between Host systems with different word lengths. It is not always clear how the sender should send data, and the receiver store it. For example, when transmitting 32-bit bytes from a 32-bit word-length system to a 36-bit word-length system, it may be desirable (for reasons of efficiency and usefulness) to store the 32-bit bytes right-justified in a 36-bit word in the latter system. In any case, the user should have the option of specifying data representation and transformation functions. It should be noted that FTP provides for very limited data type representations. Transformations desired beyond this limited capability should be performed by the user directly.

5.2 Data representation types. Data representations are handled in FTP by a user specifying a representation type. This type may implicitly (as in ASCII or EBCDIC) or explicitly (as in Local byte) define a byte size for interpretation which is referred to as the "logical byte size." This has nothing to do with the byte size used for transmission over the data connection, called the "transfer byte size", and the two should not be confused. For example, NVT-ASCII has a logical byte size of 8 bits. If the type is Local byte, then the TYPE command has an obligatory second parameter specifying the logical byte size. The transfer byte size is always 8 bits. The types ASCII and EBCDIC also take a second (optional) parameter; this is to indicate what kind of vertical format control, if any, is associated with a file. The data representation types are defined in paragraphs 5.2.1 and 5.2.2.

5.2.1 ASCII format. This is the default type and must be accepted by all FTP implementations. It is intended primarily for the transfer of text files, except when both Hosts would find the EBCDIC type more convenient. The sender converts the data from his internal character representation to the standard 8-bit NVT-ASCII representation (see the TELNET specification in MIL-STD 1782). The receiver will convert the data from the standard form to his own internal form. In accordance with the NVT standard, the <CRLF> sequence should be

MIL-STD-1780
10 May 1984

used, where necessary, to denote the end of a line of text. (See the discussion of file structure at the end of the Section on Data Representation and Storage). Using the standard NVT-ASCII representation means that data must be interpreted as 8-bit bytes.

5.2.2 EBCDIC format. This type is intended for efficient transfer between Hosts which use EBCDIC for their internal character representation. For transmission the data are represented as 8-bit EBCDIC characters. The character code is the only difference between the functional specifications of EBCDIC and ASCII types. End-of-line (as opposed to end-of-record -- see the discussion of structure) will probably be rarely used with EBCDIC type for purposes of denoting structure, but where it is necessary the <NL> character should be used.

5.3 Reasons for file transfer. A character file may be transferred to a Host for one of three purposes: for printing, for storage and later retrieval, or for processing. If a file is sent for printing, the receiving Host must know how the vertical format control is represented. In the second case, it must be possible to store a file at a Host and then retrieve it later in exactly the same form. Finally, it ought to be possible to move a file from one Host to another and process the file at the second Host without undue trouble.

5.3.1 File transfer parameters. A single ASCII or EBCDIC format does not satisfy all these conditions and so these types have a second parameter specifying one of the following three formats:

5.3.1.1 Non-print. This is the default format to be used if the second (format) parameter is omitted. Non-print format must be accepted by all FTP implementations. The file need contain no vertical format information. If it is passed to a printer process, this process may assume standard values for spacing and margins. Normally, this format will be used with files destined for processing or just storage.

5.3.1.2 TELNET format controls. The file contains ASCII/EBCDIC vertical format controls (i.e., <CR>, <LF>, <NL>, <VT>, <FF>) which the printer process will interpret appropriately. <CRLF>, in exactly this sequence, also denotes end-of-line.

5.3.1.3 Carriage control. The file contains American National Standards Institute (ANSI) (FORTRAN) vertical format control characters. In a line or a record, formatted according to the ASA Standard, the first character is not to be printed. Instead it should be used to determine the vertical movement of the paper which should take place before the rest of the record is printed. The ANSI Standard specifies the following control characters:

MIL-STD-1780
10 May 1984

Character	Vertical Spacing
blank	Move paper up one line
0	Move paper up two lines
1	Move paper to top of next page
+	No movement, i.e., overprint

Clearly there must be some way for a printer process to distinguish the end of the structural entity. If a file has record structure (see below) this is no problem; records will be explicitly marked during transfer and storage. If the file has no record structure, the <CRLF> end-of-line sequence is used to separate printing lines, but these format effectors are overridden by the ANSI controls.

5.3.1.4 Image. The data are sent as contiguous bits which, for transfer, are packed into the 8-bit transfer bytes. The receiving site must store the data as contiguous bits. The structure of the storage system might necessitate the padding of the file (or of each record, for a record-structured file) to some convenient boundary (byte, word or block). This padding, which must be all zeros, may occur only at the end of the file (or at the end of each record) and there must be a way of identifying the padding bits so that they may be stripped off if the file is retrieved. The padding transformation should be well publicized to enable a user to process a file at the storage site. Image type is intended for the efficient storage and retrieval of files and for the transfer of binary data. It is recommended that this type be accepted by all FTP implementations.

5.3.1.5 Local byte size. The data is transferred in logical bytes of the size specified by the obligatory second parameter, Byte size. The value of Byte size must be a decimal integer; there is no default value. The logical byte size is not necessarily the same as the transfer byte size. If there is a difference in byte sizes, then the logical bytes should be packed contiguously, disregarding transfer byte boundaries and with any necessary padding at the end. When the data reaches the receiving Host it will be transformed in a manner dependent on the logical byte size and the particular Host. This transformation must be invertible (that is an identical file can be retrieved if the same parameters are used) and should be well publicized by the FTP implementors.

5.3.1.5.1 Local byte example 1. A user sending 36-bit floating-point numbers to a Host with a 32-bit word could send his data as Local byte with a logical byte size of 36. The receiving Host would then be expected to store the logical bytes so that they could be easily manipulated; in this example putting the 36-bit logical bytes into 64-bit double words should suffice.

MIL-STD-1780
10 May 1984

5.3.1.5.2 Local byte example 2. A pair of hosts with a 36-bit word size may send data to one another in words by using TYPE L 36. The data would be sent in the 8-bit transmission bytes packed so that 9 transmission bytes carried two host words.

5.3.1.6 File parameter caution. A file must be stored and retrieved with the same parameters if the retrieved version is to be identical to the version originally transmitted. Conversely, FTP implementations must return a file identical to the original if the parameters used to store and retrieve a file are the same.

5.3.2 File structure. In addition to different representation types, FTP allows the structure of a file to be specified. Three file structures are defined in FTP:

file-structure, where there is no internal structure and the file is considered to be a continuous sequence of data bytes,

record-structure, where the file is made up of sequential records,

and page-structure, where the file is made up of independent indexed pages.

File-structure is the default, to be assumed if the STRUcture command has not been used but both file and record structures must be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations. The structure of a file will affect both the transfer mode of a file (see the Section on Transmission Modes) and the interpretation and storage of the file. The "natural" structure of a file will depend on which Host stores the file. A source-code file will usually be stored in fixed length records, but may also be stored as a stream of characters partitioned into lines, for example by <CRLF>. If the transfer of files between such disparate sites is to be useful, there must be some way for one site to recognize the other's assumptions about the file.

5.3.2.1 File- vs. record-oriented. With some sites being naturally file-oriented and others naturally record-oriented there may be problems if a file with one structure is sent to a Host oriented to the other. If a text file is sent with record-structure to a Host which is file-oriented, then that Host should apply an internal transformation to the file based on the record structure. Obviously this transformation should be useful but it must also be invertible so that an identical file may be retrieved using record structure. In the case of a file being sent with file-structure to a record-oriented Host, there exists the question of what criteria the Host should use to divide the file into records which can be processed locally. If this division is

MIL-STD-1780
10 May 1984

necessary the FTP implementation should use the end-of-line sequence, <CRLF> for ASCII, or <NL> for EBCDIC text files, as the delimiter. If an FTP implementation adopts this technique, it must be prepared to reverse the transformation if the file is retrieved with file-structure.

5.3.2.2 Page structure. To transmit files that are discontinuous FTP defines a page structure. Files of this type are sometimes known as "random access files" or even as "holey files". In these files there is sometimes other information associated with the file as a whole (e.g., a file descriptor), or with a section of the file (e.g., page access controls), or both. In FTP, the sections of the file are called pages. To provide for various page sizes and associated information each page is sent with a page header. The page header has the following defined fields:

- a. **Header Length.** The number of logical bytes in the page header including this byte. The minimum header length is 4.
- b. **Page Index.** The logical page number of this section of the file. This is not the transmission sequence number of this page, but the index used to identify this page of the file.
- c. **Data Length.** The number of logical bytes in the page data. The minimum data length is 0.
- d. **Page Type.** The type of page this is. The following page types are defined:

0 = Last Page

This is used to indicate the end of a paged structured transmission. The header length must be 4, and the data length must be 0.

1 = Simple Page

This is the normal type for simple paged files with no page level associated control information. The header length must be 4.

2 = Descriptor Page

This type is used to transmit the descriptive information for the file as a whole.

3 = Access Controlled Page

MIL-STD-1780
10 May 1984

This type includes an additional header field for paged files with page level access control information. The header length must be 5.

- e. Optional Fields. Further header fields may be used to supply per page control information, for example, per page access control.

All fields are one logical byte in length. The logical byte size is specified by the TYPE command.

5.4 Establishing data connections. The mechanics of transferring data consists of setting up the data connection to the appropriate ports and choosing the parameters for transfer. Both the user and the server-DTPs have a default data port. The default data connection is established between user process port U and server port L-1. The transfer byte size is 8-bit bytes. This byte size is relevant only for the actual transfer of the data; it has no bearing on representation of the data within a Host's file system.

5.4.1 Passive data transfer process. The passive data transfer process (this may be a user-DTP or a second server-DTP) shall "listen" on the data port prior to sending a transfer request command. The FTP request command determines the direction of the data transfer. The server, upon receiving the transfer request, will initiate the data connection to the port. When the connection is established, the data transfer begins between DTP's, and the server-PI sends a confirming reply to the user-PI.

5.4.2 Alternate data. It is possible for the user to specify an alternate data port by use of the PORT command. For example, he might want a file retrieved from a third party Host. In the latter case the user-PI sets up command connections with both server-PI's. One server is then told (by an FTP command) to "listen" for a connection which the other will initiate. The user-PI sends one server-PI a PORT command indicating the data port of the other. Finally both are sent the appropriate transfer commands. The exact sequence of commands and replies sent between the user-controller and the servers is defined in the Section on FTP Replies. In general, it is the server's responsibility to maintain the data connection to initiate it and to close it. The exception to this is when the user-DTP is sending the data in a transfer mode that requires the connection to be closed to indicate EOF. The server MUST close the data connection under the following conditions:

- a. The server has completed sending data in a transfer mode that requires a close to indicate EOF.
- b. The server receives an ABORT command from the user.
- c. The port specification is changed by a command from the user.

MIL-STD-1780
10 May 1984

- d. The TELNET connection is closed legally or otherwise.
- e. An irrecoverable error condition occurs.

Otherwise the close is a server option, the exercise of which he must indicate to the user-process by an appropriate reply.

5.5 Transmission modes. The next consideration in transferring data is choosing the appropriate transmission mode. There are three modes: one which formats the data and allows for restart procedures; one which also compresses the data for efficient transfer; and one which passes the data with little or no processing. In this last case the mode interacts with the structure attribute to determine the type of processing. In the compressed mode the representation type determines the filler byte. All data transfers must be completed with an end-of-file (EOF) which may be explicitly stated or implied by the closing of the data connection. For files with record structure, all the end-of-record markers (EOR) are explicit, including the final one. For files transmitted in page structure a "last-page" page type is used. For the purpose of standardized transfer, the sending Host will translate his internal end of line or end of record denotation into the representation prescribed by the transfer mode and file structure, and the receiving Host will perform the inverse translation to his internal denotation. A host specific record count field may not be recognized at another Host, so the end of record information may be transferred as a two byte control code in Stream mode or as a flagged bit in a Block or Compressed mode descriptor. End of line in an ASCII or EBCDIC file with no record structure should be indicated by <CRLF> or <NL>, respectively. Since these transformations imply extra work for some systems, identical systems transferring non-record structured text files might wish to use a binary representation and stream mode for the transfer.

NOTE: In the rest of this section, byte means "transfer byte" except where explicitly stated otherwise.

5.5.1 Transmission modes defined. The following transmission modes are defined in FTP:

5.5.1.1 Stream. The data is transmitted as a stream of bytes. There is no restriction on the representation type used; record structures are allowed. In a record structured file EOR and EOF will each be indicated by a two-byte control code. The first byte of the control code will be all ones, the escape character. The second byte will have the low order bit on and zeros elsewhere for EOR and the second low order bit on for EOF; that is, the byte will have value 1 for EOR and value 2 for EOF. EOR and EOF may be indicated together on the last byte transmitted by turning both low order bits on, i.e., the value 3. If a byte of all ones was intended to be sent as data, it should be

MIL-STD-1780
10 May 1984

repeated in the second byte of the control code. If the structure is file structure, the EOF is indicated by the sending Host closing the data connection and all bytes are data bytes.

5.5.1.2 Block. The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines: last block in the file (EOF) last block in the record (EOR), restart marker (see the Section on Error Recovery and Restart) or suspect data (i.e., the data being transferred is suspected of errors and is not reliable). This last code is NOT intended for error control within FTP. It is motivated by the desire of sites exchanging certain types of data (e.g., seismic or weather data) to send and receive all the data despite local errors (such as "magnetic tape read errors"), but to indicate in the transmission that certain portions are suspect). Record structures are allowed in this mode, and any representation type may be used. The header consists of the three bytes. Of the 24 bits of header information, the 16 low order bits shall represent byte count, and the 8 high order bits shall represent descriptor codes as shown in Figure 4.

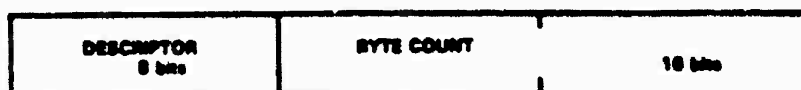


FIGURE 4. Block header.

5.5.1.2.1 Descriptor codes. The descriptor codes are indicated by bit flags in the descriptor byte. Four codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte.

Code	Meaning
128	End of data block is EOR
64	End of data block is EOF
32	Suspected errors in data block
16	Data block is a restart marker

With this encoding more than one descriptor coded condition may exist for a particular block. As many bits as necessary may be flagged. The restart marker is embedded in the data stream as an integral number of 8-bit bytes representing printable characters in the language being used over the TELNET

MIL-STD-1780
10 May 1984

connection (e.g., default—NVT-ASCII). <SP> (Space, in the appropriate language) must not be used WITHIN a restart marker. For example, Figure 5 shows the transmission of a six-character marker.

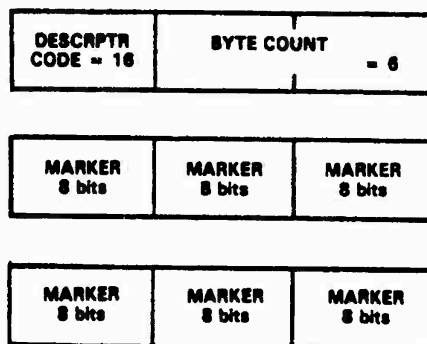
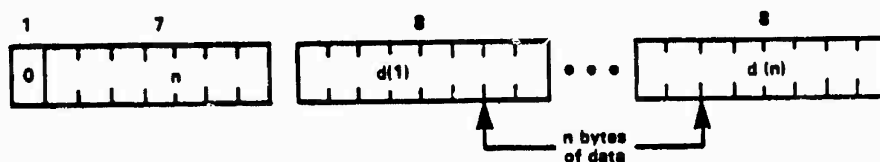


FIGURE 5. Transmission of a six-character marker.

5.5.1.3 Compressed. There are three kinds of information to be sent: regular data, sent in a byte string; compressed data, consisting of replications or filler; and control information, sent in a two-byte escape sequence. If $n > 0$ bytes (up to 127) of regular data are sent, these n bytes are preceded by a byte with the left-most bit set to 0 and the right-most 7 bits containing the number n . (See Figure 6).



String of n data bytes $d(1), \dots, d(n)$
Count n must be positive.

FIGURE 6. Byte string.

MIL-STD-1780
10 May 1984

To compress a string of n replications of the data byte d , 2 bytes are sent as shown in Figure 7.



FIGURE 7. Replicated byte.

A string of n filler bytes can be compressed into a single byte, where the filler byte varies with the representation type. If the type is ASCII or EBCDIC the filler byte is <SP> (Space, ASCII code 32., EBCDIC code 64). If the type is Image or Local byte the filler is a zero byte. (See Figure 8).

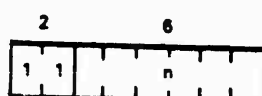


FIGURE 8. Filler string.

The escape sequence is a double byte, the first of which is the escape byte (all zeros) and the second of which contains descriptor codes as defined in Block mode. The descriptor codes have the same meaning as in Block mode and apply to the succeeding string of bytes. Compressed mode is useful for obtaining increased bandwidth on very large network transmissions at a little extra CPU cost. It can be most effectively used to reduce the size of printer files such as those generated by RJE Hosts.

5.6 Error recovery and restart. There is no provision for detecting bits lost or scrambled in data transfer; this level of error control is handled by the TCP. However, a restart procedure is provided to protect users from gross system failures (including failures of a Host, an FTP-process, or the underlying network). The restart procedure is defined only for the block and compressed modes of data transfer. It requires the sender of data to insert a special marker code in the data stream with some marker information. The marker information has meaning only to the sender, but must consist of printable characters in the default or negotiated language of the TELNET connection (ASCII or EBCDIC). The marker could represent a bit-count, a record-count, or any other information by which a system may identify a data checkpoint. The receiver of data, if it implements the restart procedure, would then mark the corresponding position of this marker in the receiving system, and return this information to the user. In the event of a system failure, the user can restart the data transfer by identifying the marker

MIL-STD-1780
10 May 1984

point with the FTP restart procedure. The following example illustrates the use of the restart procedure. The sender of the data inserts an appropriate marker block in the data stream at a convenient point. The receiving Host marks the corresponding data point in its file system and conveys the last known sender and receiver marker information to the user, either directly or over the command connection in a 110 reply (depending on who is the sender). In the event of a system failure, the user or controller process restarts the server at the last server marker by sending a restart command with server's marker code as its argument. The restart command is transmitted over the command connection and is immediately followed by the command (such as RETR, STOR or LIST) which was being executed when the system failure occurred.

5.7 File transfer functions. The communication channel from the user-PI to the server-PI is established by a TCP connection from the user to a standard server port. The user protocol interpreter is responsible for sending FTP commands and interpreting the replies received; the server-PI interprets commands, sends replies and directs its DTP to set up the data connection and transfer the data. If the second party to the data transfer (the passive transfer process) is the user-DTP then it is governed through the internal protocol of the user-FTP Host; if it is a second server-DTP then it is governed by its PI on command from the user-PI. The FTP replies are discussed in the next section. In the description of a few of the commands in this section it is helpful to be explicit about the possible replies.

5.7.1 FTP commands. The following are FTP commands:

5.7.1.1 Access control commands. The following commands specify access control identifiers (command codes are shown in parentheses).

5.7.1.1.1 User name (USER). The argument field is a TELNET string identifying the user. The user identification is that which is required by the server for access to its file system. This command will normally be the first command transmitted by the user after the command connections are made (some servers may require this). Additional identification information in the form of a password and/or an account command may also be required by some servers. Servers may allow a new USER command to be entered at any point in order to change the access control and/or accounting information. This has the effect of flushing any user, password, and account information already supplied and beginning the login sequence again. All transfer parameters are unchanged and any file transfer in progress is completed under the old account.

5.7.1.1.2 Password (PASS). The argument field is a TELNET string identifying the user's password. This command must be immediately preceded by the user name command, and, for some sites, completes the user's identification

MIL-STD-1780
10 May 1984

for access control. Since password information is quite sensitive, it is desirable in general to "mask" it or suppress typeout. It appears that the server has no foolproof way to achieve this. It is therefore the responsibility of the user-FTP process to hide the sensitive password information.

5.7.1.1.3 Account (ACCT). The argument field is a TELNET string identifying the user's account. The command is not necessarily related to the USER command, as some sites may require an account for login and others only for specific access, such as storing files. In the latter case the command may arrive at any time. There are reply codes to differentiate these cases for the automaton: when account information is required for login, the response to a successful PASSword command is reply code 332. On the other hand, if account information is NOT required for login, the reply to a successful PASSword command is 230; and if the account information is needed for a command issued later in the dialogue, the server should return a 332 or 532 reply depending on whether he stores (pending receipt of the ACCount command) or discards the command, respectively.

5.7.1.1.4 Reinitialize (REIN). This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the command connection is left open. This is identical to the state in which a user finds himself immediately after the command connection is opened. A USER command may be expected to follow.

5.7.1.1.5 Logout (QUIT). This command terminates a USER and if file transfer is not in progress, the server closes the command connection. If file transfer is in progress, the connection will remain open for result response and the server will then close it. If the user-process is transferring files for several USERS but does not wish to close and then reopen connections for each, then the REIN command should be used instead of QUIT. An unexpected close on the command connection will cause the server to take the effective action of an abort (ABOR) and a logout (QUIT).

5.7.2 Transfer parameter commands. All data transfer parameters have default values, and the commands specifying data transfer parameters are required only if the default parameter values are to be changed. The default value is the last specified value, or if no value has been specified, the standard default value as stated here. This implies that the server must "remember" the applicable default values. The commands may be in any order except that they must precede the FTP service request. The following commands specify data transfer parameters.

MIL-STD-1780
10 May 1984

5.7.2.1 Data port (PORT). The argument is a HOST-PORT specification for the data port to be used in data connection. There defaults for both the user and server data ports, and under normal circumstances this command and its reply are not needed. If this command is used the argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each

field is transmitted as a decimal number (in character string representation). The fields are separated by commas. A port command would be PORT h1,h2,h3,h4, p1,p2; where, h1 is the high order 8 bits of the internet host address.

5.7.2.2 Passive (PASV). This command requests the server-DTP to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command. The response to this command includes the host and port address this server is listening on.

5.7.2.3 Representation type (TYPE). The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter is denoted by a single TELNET character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32.). The following codes are assigned for type:

A - ASCII	} -> {	N - Non-print
E - EBCDIC		T - TELNET format effectors
i - Image		C - Carriage Control (ASA)

L <byte size> - Local byte Byte size

The default representation type is ASCII Non-print. If the Format parameter is changed, and later just the first argument is changed, Format then returns to the Non-print default.

5.7.2.4 File structure (STRU). The argument is a single TELNET character code specifying file structure described in the Section on Data Representation and Storage. The following codes are assigned for structure:

F - File (no record structure)
R - Record structure
P - Page structure

The default structure is File.

MIL-STD-1780

10 May 1984

5.7.2.5 Transfer mode (MODE). The argument is a single TELNET character code specifying the data transfer modes described in the Section on Transmission Modes. The following codes are assigned for transfer modes:

- S - Stream
- B - Block
- C - Compressed

The default transfer mode is Stream.

5.7.3 FTP service commands. The FTP service commands define the file transfer or the file system function requested by the user. The argument of an FTP service command will normally be a pathname. The syntax of pathnames must conform to server site conventions (with standard defaults applicable), and the language conventions of the command connection. The suggested default handling is to use the last specified device, directory or file name, or the standard default defined for local users. The commands may be in any order except that a "rename from" command must be followed by a "rename to" command and the restart command must be followed by the interrupted service command. The data, when transferred in response to FTP service commands, shall always be sent over the data connection, except for certain informative replies. The following commands specify FTP service requests:

5.7.3.1 Retrieve (RETR). This command causes the server-DTP to transfer a copy of the file, specified in the pathname, to the server user-DTP at the other end of the data connection. The status and contents of the file at the server site shall be unaffected.

5.7.3.2 Store (STOR). This command causes the server-DTP to accept the data transferred via the data connection and to store the data as a file at the server site. If the file specified in the pathname exists at the server site then its contents shall be replaced by the data being transferred. A new file is created at the server site if the file specified in the pathname does not already exist.

5.7.3.3 Append (with create) (APPE). This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, then the data shall be appended to that file; otherwise the file specified in the pathname shall be created at the server site.

5.7.3.4 Allocate (ALLO). This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred. The argument shall be a decimal integer representing the number of bytes (using the logical byte size) of storage to be reserved for the file. For files sent with record or page structure a maximum record or page size (in logical bytes) might also be necessary; this is indicated by a decimal integer in a second

MIL-STD-1780
10 May 1984

argument field of the command. This second argument is optional, but when present should be separated from the first by the three TELNET characters <SP> R <SP>. This command shall be followed by a STORe or APPEnd command. The ALLO command should be treated as a NOOP (no operation) by those servers which do not require that the maximum size of the file be declared beforehand, and those servers interested in only the maximum record or page size should accept a dummy value in the first argument and ignore it.

5.7.3.5 Restart (REST). The argument field represents the server marker at which file transfer is to be restarted. This command does not cause file transfer but "spaces" over the file to the specified data checkpoint. This command shall be immediately followed by the appropriate FTP service command which shall cause file transfer to resume.

5.7.3.6 Rename from (RNFR). This command specifies the file which is to be renamed. This command must be immediately followed by a "rename to" command specifying the new file pathname.

5.7.3.7 Rename to (RNTO). This command specifies the new pathname of the file specified in the immediately preceding "rename from" command. Together the two commands cause a file to be renamed.

5.7.3.8 Abort (ABOR). This command tells the server to abort the previous FTP service command and any associated transfer of data. The abort command may require "special action", as discussed in the Section on FTP Commands, to force recognition by the server. No action is to be taken if the previous command has been completed (including data transfer). The TELNET connection is not to be closed by the server, but the data connection must be closed. There are two cases for the server upon receipt of this command: (1) the FTP service command was already completed, or (2) the FTP service command is still in progress.

5.7.3.8.1 FTP service command completed. In the first case, the server closes the data connection (if it is open) and responds with a 226 reply, indicating that the abort command was successfully processed.

5.7.3.8.2 FTP service command in progress. In the second case, the server aborts the FTP service in progress and closes the data connection, returning a 426 reply to indicate that the service request terminated in abnormally. The server then sends a 226 reply, indicating that the abort command was successfully processed.

5.7.3.9 Delete (DELE). This command causes the file specified in the pathname to be deleted at the server site. If an extra level of protection is desired (such as the query, "DO you really wish to delete?"), it should be provided by the user-FTP process.

MIL-STD-1780
10 May 1984

5.7.3.10 Change working directory (CWD). This command allows the user to work with a different directory or dataset for file storage or retrieval without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

5.7.3.11 List (LIST). This command causes a list to be sent from the server to the passive DTP. If the pathname specifies a directory, the server should transfer a list of files in the specified directory. If the pathname specifies a file then the server should send current information on the file. A null argument implies the user's current working or default directory. The data transfer is over the data connection in type ASCII or type EBCDIC. (The user must ensure that the TYPE is appropriately ASCII or EBCDIC).

5.7.3.12 Name-list (NLST). This command causes a directory listing to be sent from server to user site. The pathname should specify a directory or other system-specific file group descriptor; a null argument implies the current directory. The server will return a stream of names of files and no other information. The data will be transferred in ASCII or EBCDIC type over the data connection as valid pathname strings separated by <CRLF> or <NL>. (Again the user must ensure that the TYPE is correct.)

5.7.3.13 Site parameters (SITE). This command is used by the server to provide services specific to his system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol. The nature of these services and the specification of their syntax can be stated in a reply to the HELP SITE command.

5.7.3.14 Status (STAT). This command shall cause a status response to be sent over the command connection in the form of a reply. The command may be sent during a file transfer (along with the command IP and Synch signals--see the Section on FTP Commands) in which case the server will respond with the status of the operation in progress, or it may be sent between file transfers. In the latter case the command may have an argument field. If the argument is a pathname, the command is analogous to the "list" command except that data shall be transferred over the command connection. If a partial pathname is given, the server may respond with a list of file names or attributes associated with that specification. If no argument is given, the server should return general status information about the server FTP process. This should include current values of all transfer parameters and the status of connections.

5.7.3.15 Help (HELP). This command shall cause the server to send helpful information regarding its implementation status over the command connection to the user. The command may take an argument (e.g., any command name) and

MIL-STD-1780
10 May 1984

return more specific information as a response. The reply is type 211 or 214. It is suggested that HELP be allowed before entering a USER command. The server may use this reply to specify site-dependent parameters, e.g., in response to HELP SITE.

5.7.3.16 Noop (NOOP). This command does not affect any parameters or previously entered commands. It specifies no action other than that the server send an OK reply.

5.7.4 TELNET language. The File Transfer Protocol follows the specifications of the TELNET protocol for all communications over the command connection. Since the language used for TELNET communication may be a negotiated option, all references in the next two sections will be to the "TELNET language" and the corresponding "TELNET end of line code". Currently one may take these to mean NVT-ASCII and <CR><LF>. No other specifications of the TELNET protocol will be cited. FTP commands are "TELNET strings" terminated by the "TELNET end of line code". The command codes themselves are alphabetic characters terminated by the character <SP> (Space) if parameters follow and TELNET-EOL otherwise. The command codes and the semantics of commands are described in this section; the detailed syntax of commands is specified in the Section on Commands, the reply sequences are discussed in the Section on Sequencing of Commands and Replies, and scenarios illustrating the use of commands are provided in the Section on Typical FTP Scenarios. FTP commands may be partitioned as those specifying access-control identifiers, data transfer parameters, or FTP service requests. Certain commands (such as ABOR, STAT, QUIT) may be sent over the command connection while a data transfer is in progress. Some servers may not be able to monitor the command and data connections simultaneously, in which case some special action will be necessary to get the server's attention. The exact form of the "special action" is undefined; but the following ordered format is tentatively recommended:

- a. User system inserts the TELNET "Interrupt Process" (IP) signal in the TELNET stream.
- b. User system sends the TELNET "Synch" signal
- c. User system inserts the command (e.g., ABOR) in the TELNET stream.
- d. Server PI, after receiving "IP", scans the TELNET stream for EXACTLY ONE FTP command.

(For other servers this may not be necessary but the actions listed above should have no unusual effect.)

5.8 FTP replies. Replies to File Transfer Protocol commands are devised to ensure the synchronization of requests and actions in the process of file transfer, and to guarantee that the user process always knows the state of the

MIL-STD-1780

10 May 1984

Server. Every command must generate at least one reply, although there may be more than one; in the latter case, the multiple replies must be easily distinguished. In addition, some commands occur in sequential groups, such as USER, PASS and ACCT, or RNFR and RNTD. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning. The details of the command-reply sequence are made explicit in a set of state diagrams below. An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intended that the three digits contain enough encoded information that the user-process (the User-PI) will not need to examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be server-dependent, so there are likely to be varying texts for each reply code.

5.8.1 FTP reply defined. Formally, a reply is defined to contain the 3-digit code, followed by Space <SP>, followed by one line of text (where some maximum line length has been specified), and terminated by the TELNET end-of-line code. There will be cases, however, where the text is longer than a single line. In these cases the complete text must be bracketed so the User-process knows when it may stop reading the reply (i.e. stop processing input on the command connection) and go do other things. This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to indicate the state of the transaction. To satisfy all factions it was decided that both the first and last line codes should be the same. Thus the format for multi-line replies is that the first line will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus), followed by text. The last line will begin with the same code, followed immediately by a Space <SP>, optionally some text, and the TELNET end-of-line code. For example:

```
123-First line
Second line
234 A line beginning with numbers
123 The last line
```

The user-process then simply needs to search for the second occurrence of the same reply code, followed by <SP> (Space), at the beginning of a line, and ignore all intermediary lines. If an intermediary line begins with a 3-digit number, the Server must pad the front to avoid confusion. This scheme allows standard system routines to be used for reply information (such as for the STAT reply), with "artificial" first and last lines tacked on. In the rare cases where these routines are able to generate three digits and a Space at the beginning of any line, the beginning of each text line should be offset by

MIL-STD-1780
10 May 1984

some neutral text, like Space. This scheme assumes that multi-line replies may not be nested. We have found that, in general, nesting of replies will not occur, except for random system messages (also called spontaneous replies) which may interrupt another reply. System messages (i.e. those not processed by the FTP server) will NOT carry reply codes and may occur anywhere in the command-reply sequence. They may be ignored by the User-process as they are only information for the human user.

5.8.2 Three-digit reply defined. The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated response by the user-process. The first digit denotes whether the response is good, bad or incomplete. (Referring to the state diagram) an unsophisticated user-process will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A user-process that wants to know approximately what kind of error occurred (e.g. file system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information (e.g. RNT0 command without a preceding RNFR.)

5.8.2.1 First digit values. There are five values for the first digit of the reply code:

5.8.2.1.1 Positive preliminary reply (1yz). The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult.

5.8.2.1.2 Positive completion reply (2yz). The requested action has been successfully completed. A new request may be initiated.

5.8.2.1.3 Positive intermediate reply (3yz). The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

5.8.2.1.4 Transient negative completion reply (4yz). The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that

MIL-STD-1780
10 May 1984

the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g. the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5.8.2.1.5 Permanent negative completion reply (5yz). The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g. after the spelling has been changed, or the user has altered his directory status.)

5.8.2.2 Second digit values. The following function groupings are encoded in the second digit:

5.8.2.2.1 Syntax (x0z). These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.

5.8.2.2.2 Information (x1z). These are replies to requests for information, such as status or help.

5.8.2.2.3 Connections (x2z). Replies referring to the command and data connections.

5.8.2.2.4 Authentication and accounting (x3z). Replies for the login process and accounting procedures.

5.8.2.2.5 Unspecified (x4z).

5.8.2.2.6 File system (x5z). These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

5.8.2.3 Third digit values. The third digit gives a finer gradation of meaning in each of the function categories, specified by the second digit. The list of replies below will illustrate this. Note that the text associated with each reply is recommended, rather than mandatory, and may even change according to the command with which it is associated. The reply codes, on the other hand, must strictly follow the specifications in the last section; that is, Server implementations should not invent new codes for situations that are only slightly different from the ones described here, but rather should adapt codes already defined. A command such as TYPE or ALLO whose successful execution does not offer the user-process any new information will cause a 200

MIL-STD-1780
10 May 1984

reply to be returned. If the command is not implemented by a particular Server-FTP process because it has no relevance to that computer system, for example ALLO at a TOPS20 site, a Positive Completion reply is still desired so that the simple User-process knows it can proceed with its course of action. A 202 reply is used in this case with, for example, the reply text: "No storage allocation necessary." If, on the other hand, the command requests a non-site-specific action and is unimplemented, the response is 502. A refinement of that is the 504 reply for a command that IS implemented, but that requests an unimplemented parameter.

5.8.3 Reply codes by function groups.

- a. 200 Command okay.
- b. 500 Syntax error, command unrecognized.
[This may include errors such as command line too long.]
- c. 501 Syntax error in parameters or arguments.
- d. 202 Command not implemented, superfluous at this site.
- e. 502 Command not implemented.
- f. 503 Bad sequence of commands.
- g. 504 Command not implemented for that parameter.
- h. 110 Restart marker reply.
In this case the text is exact and not left to the particular implementation; it must read:
MARK yyyy = mmmmm
where yyyy is User-process data stream marker, and mmmmm server's equivalent marker. (Note the spaces between markers and "=".)
- i. 119 Terminal not available, will try mailbox.
- j. 211 System status, or system help reply.
- k. 212 Directory status.
- l. 213 File status.
- m. 214 Help message.
(On how to use the server or the meaning of a particular nonstandard command. This reply is useful only to the human user.)

MIL-STD-1780
10 May 1984

- n. 215 <scheme> is the preferred scheme.
- o. 120 Service ready in nnn minutes.
- p. 220 Service ready for new user.
- q. 221 Service closing TELNET connection.
(logged out if appropriate)
- r. 421 Service not available, closing TELNET connection.
[This may be a reply to any command if the service knows it must shut down.]
- s. 125 Data connection already open; transfer starting.
- t. 225 Data connection open; no transfer in progress.
- u. 425 Can't open data connection.
- v. 226 Closing data connection; requested file action successful.
(For example, file transfer or file abort.)
- w. 426 Connection closed; transfer aborted.
- x. 227 Entering Passive Mode. h1,h2,h3,h4,p1,p2.
- y. 230 User logged in, proceed.
- z. 530 Not logged in.
- aa. 331 User name okay, need password.
- bb. 332 Need account for login.
- cc. 532 Need account for storing files.
- dd. 150 File status okay; about to open data connection.
- ee. 151 User not local; Will forward to <user>@<host>.
- ff. 152 User Unknown; Mail will be forwarded by the operator.
- gg. 250 Requested file action okay, completed.
- hh. 350 Requested file action pending further information.

MIL-STD-1780

10 May 1984

- ii. 450 Requested file action not taken: file unavailable (e.g. file busy).
- jj. 550 Requested action not taken: file unavailable (e.g. file not found, no access).
- kk. 451 Requested action aborted: local error in processing.
- ll. 551 Requested action aborted: page type unknown.
- mm. 452 Requested action not taken: insufficient storage space in system
- nn. 552 Requested file action aborted: exceeded storage allocation. (for current directory or dataset)
- oo. 553 Requested action not taken: file name not allowed
- pp. 354 Start mail input; end with <CR><LF>.<CR><LF>.

5.8.4 Numeric order list of reply codes.

- a. 110 Restart marker reply.
In this case the text is exact and not left to the particular implementation; it must read:
MARK yyyy = mmmm
where yyyy is User-process data stream marker, and mmmm server's equivalent marker. (note the spaces between markers and "=".)
- b. 119 Terminal not available, will try mailbox.
- c. 120 Service ready in nnn minutes.
- d. 125 Data connection already open; transfer starting.
- e. 150 File status okay; about to open data connection.
- f. 151 User not local; Will forward to <user>@<host>.
- g. 152 User Unknown; Mail will be forwarded by the operator.
- h. 200 Command okay.
- i. 202 Command not implemented, superfluous at this site.
- j. 211 System status, or system help reply.

MIL-STD-1780
10 May 1984

- k. 212 Directory status.
- l. 213 File status.
- m. 214 Help message.
(On how to use the server or the meaning of a particular nonstandard command. This reply is useful only to the human user.)
- n. 215 <scheme> is the preferred scheme.
- o. 220 Service ready for new user.
- p. 221 Service closing TELNET connection.
(Logged out if appropriate.)
- q. 225 Data connection open; no transfer in progress.
- r. 226 Closing data connection; requested file action successful.
(For example, file transfer or file abort.)
- s. 227 Entering Passive Mode. h1,h2,h3,h4,p1,p2
- t. 230 User logged in, proceed.
- u. 250 Requested file action okay, completed.
- v. 331 User name okay, need password.
- w. 332 Need account for login.
- x. 350 Requested file action pending further information.
- y. 354 Start mail input; end with <CR><LF>.<CR><LF>.
- z. 421 Service not available, closing TELNET connection.
[This may be a reply to any command if the service knows it must shut down.]
- aa. 425 Can't open data connection.
- bb. 426 Connection closed; transfer aborted.
- cc. 450 Requested file action not taken: file unavailable (e.g. file busy).
- dd. 451 Requested action aborted: local error in processing.

MIL-STD-1780
10 May 1984

- ee. 452 Requested action not taken: insufficient storage space in system.
- ff. 500 Syntax error, command unrecognized.
[This may include errors such as command line too long.]
- gg. 501 Syntax error in parameters or arguments.
- hh. 502 Command not implemented.
- ii. 503 Bad sequence of commands.
- jj. 504 Command not implemented for that parameter.
- kk. 530 Not logged in.
- ll. 532 Need account for storing files.
- mm. 550 Requested action not taken: file unavailable (e.g. file not found, no access).
- nn. 551 Requested action aborted: page type unknown.
- oo. 552 Requested file action aborted: exceeded storage allocation (for current directory or dataset).
- pp. 553 Requested action not taken: file name not allowed.

5.9 Declarative specifications.

5.9.1 Minimum implementation. In order to make FTP workable without need-less error messages, the following minimum implementation is required for all servers:

TYPE - ASCII Non-print
MODE - Stream
STRUCTURE - File, Record
COMMANDS - USER, QUIT, PORT,
TYPE, MODE, STRU,
for the default values
RETR, STOR,
NOOP.

MIL-STD-1780
10 May 1984

The default values for transfer parameters are:

TYPE - ASCII Non-print
MODE - Stream
STRU - File

All Hosts must accept the above as the standard defaults.

5.9.2 Connections. The server protocol interpreter shall "listen" on Port L. The user or user protocol interpreter shall initiate the full-duplex command connection. Server and user processes should follow the conventions of the TELNET protocol as specified in MIL-STD 1782, TELNET Protocol. Servers are under no obligation to provide for editing of command lines and may specify that it be done in the user Host. The command connection shall be closed by the server at the user's request after all transfers and replies are completed. The user-DTP must "listen" on the specified data port; this may be the default user port (U) or a port specified in the PORT command. The server shall initiate the data connection from his own default data port (L-1) using the specified user data port. The direction of the transfer and the port used will be determined by the FTP service command.

5.9.2.1 Command-reply sequence. When data is to be transferred between two servers, A and B (refer to Figure 2), the user-PI, C, sets up command connections with both server-PI's. One of the servers, say A, is then sent a PASV command telling him to "listen" on his data port rather than initiate a connection when he receives a transfer service command. When the user-PI receives an acknowledgment to the PASV command, which includes the identity of the host and port being listened on, the user-PI then sends A's port, a, to B in a PORT command; a reply is returned. The user-PI may then send the corresponding service commands to A and B. Server B initiates the connection and the transfer proceeds. The command-reply sequence is listed below where the messages are vertically synchronous but horizontally asynchronous:

User-PI - Server A

C->A : Connect
C->A : PASV
A->C : 227 Entering Passive Mode. A1,A2,A3,A4,a1,a2
C-> : STOR
B->A : Connect to HOST-A, PORT-a

User-PI - Server B

C->B : Connect
C->B : PORT A1,A2,A3,A4,a1,a2
B->C : 200 Okay
C->B : RETR

The data connection shall be closed by the server under the conditions described in the Section on Establishing Data Connections. If the server

MIL-STD-1780
10 May 1984

wishes to close the connection after a transfer where it is not required, he should do so immediately after the file transfer is completed. He should not wait until after a new transfer command is received because the user-process will have already tested the data connection to see if it needs to do a "listen"; (recall that the user must "listen" on a closed data port BEFORE sending the transfer request). To prevent a race condition here, the server sends a reply (226) after closing the data connection (or if the connection is left open, a "file transfer completed" reply (250) and the user-PI should wait for one of these replies before issuing a new transfer command.

5.9.3 Commands. The commands are TELNET character string transmitted over the command connections as described in the Section on FTP Commands. The command functions and semantics are described in the Section on Access Control Commands, Transfer Parameter Commands, FTP Service Commands, and Miscellaneous Commands. The command syntax is specified here. The commands begin with a command code followed by an argument field. The command codes are four or fewer alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus, RETR, Retr, retr, ReTr, or rETr may represent the retrieve command. This also applies to any symbols representing parameter values, such as A or a for ASCII TYPE. The command codes and the argument fields are separated by one or more spaces. The argument field consists of a variable length character string ending with the character sequence <CRLF> (Carriage Return, Linefeed) for NVT-ASCII representation; for other negotiated languages a different end of line character might be used. It should be noted that the server is to take NO action until the end of line code is received. The syntax is specified below in NVT-ASCII. All characters in the argument field are ASCII characters including any ASCII represented decimal integers. Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

5.9.3.1 FTP command list. The following are the FTP commands:

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <account information> <CRLF>
REIN <CRLF>
QUIT <CRLF>
PORT <SP> <Host-port> <CRLF>
PASV <CRLF>
TYPE <SP> <type code> <CRLF>
STRU <SP> <structure code> <CRLF>
MODE <SP> <mode code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
APPE <SP> <pathname> <CRLF>
ALLO <SP> <decimal integer>
      [<SP> R <SP> <decimal integer>] <CRLF>
```

MIL-STD-1780
10 May 1984

```

REST <SP> <marker> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNT0 <SP> <pathname> <CRLF>
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
CWD <SP> <pathname> <CRLF>
LIST [<SP> <pathname>] <CRLF>
NLST [<SP> <pathname>] <CRLF>
SITE <SP> <string> <CRLF>
STAT [<SP> <pathname>] <CRLF>
HELP [<SP> <string>] <CRLF>
NOOP <CRLF>

```

5.9.3.2 FTP command syntax. The syntax of the above argument fields (using BNF notation where applicable) is:

```

<username> ::= <string>
<password> ::= <string>
<account information> ::= <string>

<string> ::= <char> <char><string>
<char> ::= any of the 128 ASCII characters except <CR> and <LF>
<marker> ::= <pr string>
<pr string> ::= <pr char> <pr char><pr string>
<pr char> ::= printable characters, any
               ASCII code 33 through 126
<byte size> ::= any decimal integer 1 through 255
<Host-port> ::= <Host-number>,<Port-number>
<Host-number> ::= <number>,<number>,<number>,<number>
<Port-number> ::= <number>,<number>
<number> ::= any decimal integer 0 through 255
<ident> ::= <string>
<scheme> ::= R T ?
<form code> ::= N T C
<type code> ::= A [<SP> <form code>]
               E [<SP> <form code>]
               I
               L <SP> <byte size>
<structure code> ::= F R P
<mode code> ::= S B C
<pathname> ::= <string>

```

5.10 Sequencing of commands and replies. The communication between the user and server is intended to be an alternating dialogue. As such, the user issues an FTP command and the server responds with a prompt primary reply.

MIL-STD-1780
10 May 1984

Logout
QUIT
221
500
REIN
120
220
220
421
500, 502
Transfer parameters
PORT
200
500, 501, 421, 530
PASV
227
500, 501, 502, 421, 530
MODE, TYPE, STRU
200
500, 501, 504, 421, 530
File action commands
ALLO
200
202
500, 501, 504, 421, 530
REST
500, 501, 502, 421, 530
350

STOR
125, 150
(110)
226, 250
425, 426, 451, 551, 552
532, 450, 452, 553
500, 501, 421, 530

RETR
125, 150
(110)
226, 250
425, 426, 451
450, 550
500, 501, 421, 530

MIL-STD-1780
10 May 1984

The user should wait for this initial primary success or failure response before sending further commands. Certain commands require a second reply for which the user should also wait. These replies may, for example, report on the progress or completion of file transfer or the closing of the data connection. They are secondary replies to file transfer commands. One important group of informational replies is the connection greetings. Under normal circumstances, a server will send a 220 reply, "awaiting input", when the connection is completed. The user should wait for this greeting message before sending any commands. If the server is unable to accept input right away, he should send a 120 "expected delay" reply immediately and a 220 reply when ready. The user will then know not to hang up if there is a delay. The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a server may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

5.10.1 Command-reply sequences. In this section, the command-reply sequence is presented. Each command is listed with its possible replies; command groups are listed together. Preliminary replies are listed first (with their succeeding replies indented and under them), then positive and negative completion, and finally intermediary replies with the remaining commands from the sequence following. This listing forms the basis for the state diagrams, which will be presented separately.

Connection Establishment

120

220

220

421

Login

USER

230

530

500, 501, 421

331, 332

PASS

230

202

530

500, 501, 503, 421

332

ACCT

230

202

530

500, 501, 503, 421

MIL-STD-1780
10 May 1984

LIST, NLST
125, 150
226, 250
425, 426, 451
450
500, 501, 502, 421, 530

APPE
125, 150
(110)
226, 250
425, 426, 451, 551, 552
532, 450, 550, 452, 553
500, 501, 502, 421, 530

RNFR
450, 550
500, 501, 502, 421, 530
350

RNTD
250
532, 553
500, 501, 502, 503, 421, 530

DELE, CMD
250
450, 550
500, 501, 502, 421, 530

ABOR
225, 226
500, 501, 502, 421
Informational commands

STAT
211, 212, 213
450
500, 501, 502, 421, 530

HELP
211, 214
500, 501, 502, 421

Miscellaneous commands

SITE
200
202
500, 501, 530

NOOP
200
500 421

MIL-STD-1780
10 May 1984

5.11 State diagrams. Here are presented state diagrams for a very simple FTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of FTP commands or command sequences. The command groupings were determined by constructing a model for each command then collecting together the commands with structurally identical models. For each command or command sequence there are three possible outcomes: success (S), failure (F), and error (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

Figure 9 represents the largest group of FTP commands.

5.11.1 Largest group of FTP commands.

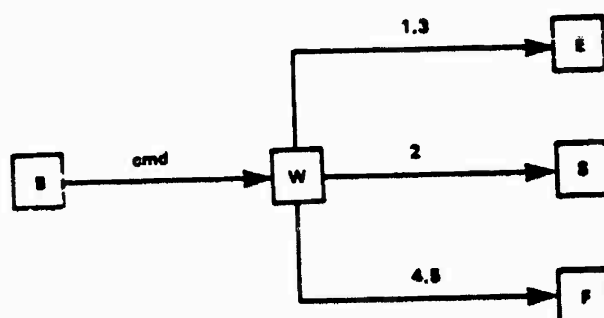


FIGURE 9. Largest group of FTP commands.

Figure 9 models the commands ABOR, ALLO, DELE, CWD, HELP, MODE, MRCP, MRCP, MRSQ, NOOP, PASV, QUIT, SITE, PORT, STAT, STRU and TYPE.

5.11.2 Other large group of FTP commands. The other large group of commands is represented by a very similar diagram.

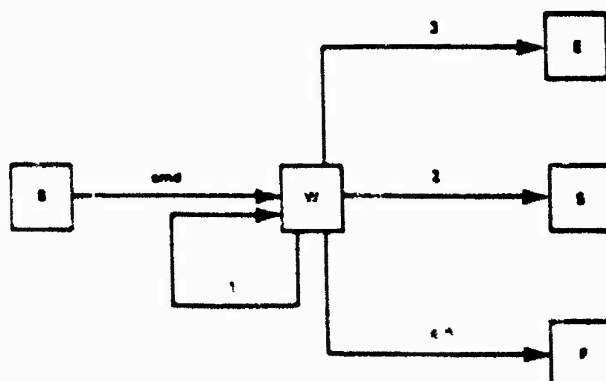


FIGURE 10. Other group of FTP commands.

MIL-STD-1780
10 May 1984

Figure 10 models the commands APPE, LIST, MLFL, NLST, REIN, RETR, STOR. Note that this second model could also be used to represent the first group of commands, the only difference being that in the first group the 100 series replies are unexpected and therefore treated as error, while the second group expects (some may require) 100 series replies.

5.11.3 Rename sequence command. The remaining diagrams model command sequences, perhaps the simplest of these is the rename sequence shown in Figure 11.

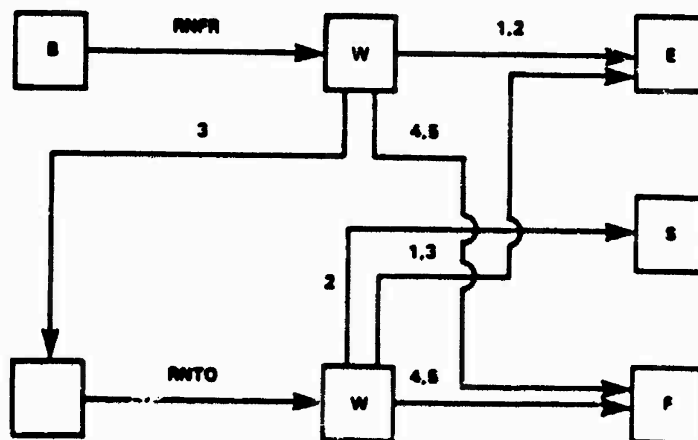


FIGURE 11. Rename sequence.

MIL-STD-1780
10 May 1984

5.11.4 Restart command. Figure 12 is a simple model of the Restart command where "cmd" is APPE, STOR, or RETR. Note that the Restart and Rename models are similar. The Restart differs only in the treatment of 100 series replies at the second stage.

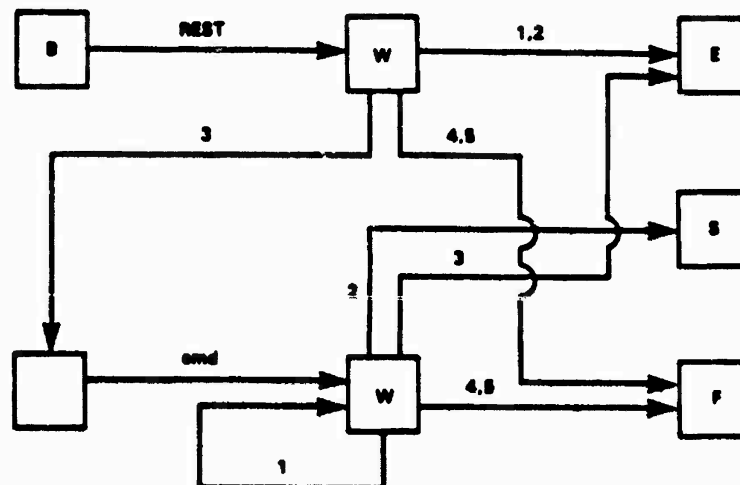
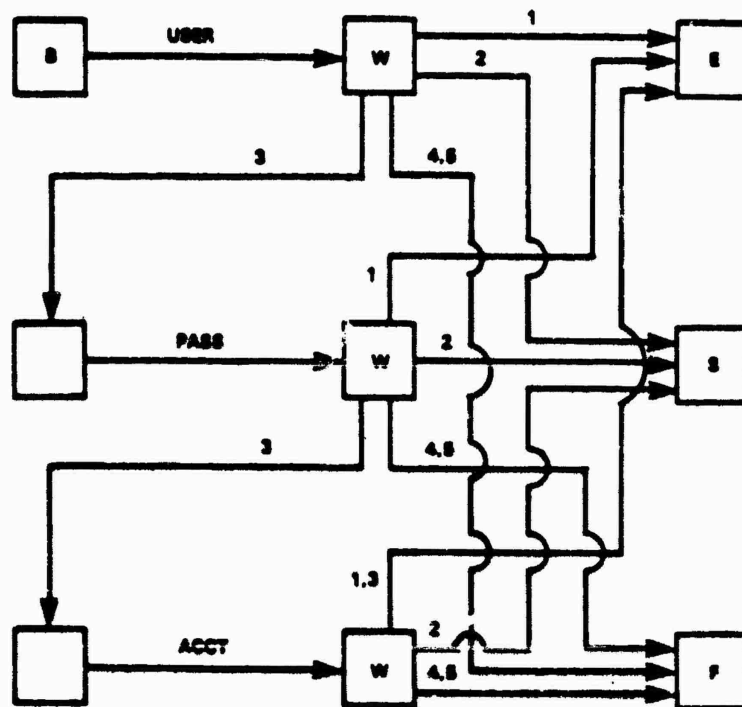


FIGURE 12. Restart command.

MIL-STD-1780
10 May 19845.11.5 login sequence. The most complicated diagram is for login sequence:FIGURE 13. login sequence

MIL-STD-1780
10 May 1984

5.11.6 Command and reply interchange. Finally, is presented a generalized diagram that could be used to model the command and reply interchange:

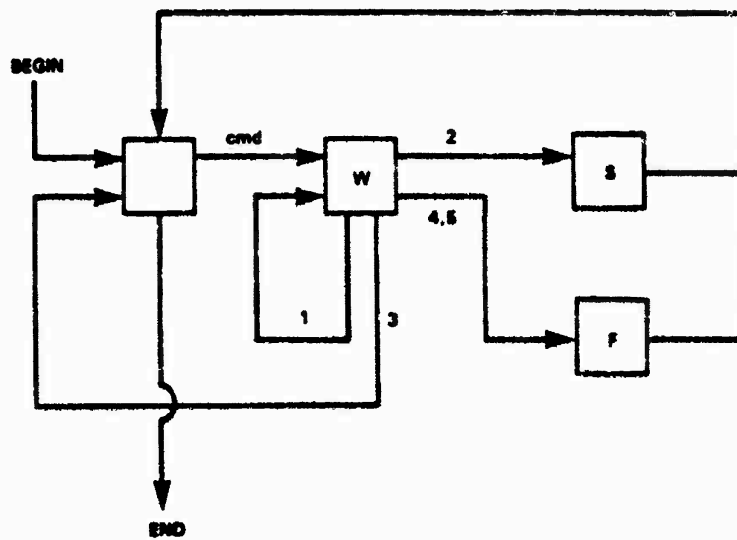


FIGURE 14. Command and reply interchange.

MIL-STD-1780
10 May 1984

6. TYPICAL FTP SCENARIO

6.1 Scenario. User at Host U wanting to transfer files to/from Host S. In general the user will communicate to the server via a mediating user-FTP process. The following may be a typical scenario. The user-FTP prompts are shown in parentheses, '——>' represents commands from Host U to Host S, and '<——' represents replies from Host S to Host U.

LOCAL COMMANDS BY USER	ACTION INVOLVED
ftp (host) opsys<CR>	Connect to Host S, port L, establishing command connections <—— 220 Service ready <CRLF>
username Doe <CR>	USER Doe<CRLF>——> <—— 331 User name ok, need password<CRLF>
password pswd <CR>	PASS mumble<CRLF>——> <—— 230 User logged in.<CRLF>
retrieve (local type) ASCII<CR> (local pathname) test 1 <CR> (for.pathname) test.pll<CR>	User-FTP opens local file in ASCII. RETR test.pll<CRLF> ——> <—— 150 File status okay; about to open data connection Server makes data connection to port U
<CRLF>	<—— 226 Closing data connection, file transfer successful<CRLF>
type Image<CR>	TYPE I<CRLF> —— <—— 200 Command OK<CRLF>
store (local type) image<CR> (local pathname) file dump<CR> (for.pathname) >udd>cn>fd<CR>	User-FTP opens local file in Image. STOR >udd>cn>fd<CRLF> ——> <—— 450 Access denied<CRLF>
terminate	QUIT <CRLF> ——> Server closes all connections.

6.2 Connection establishment. The FTP command connection is established via TCP between the user process port U and the server process port L. This protocol is assigned the service port 21 (25 octal), that is L=21.

Custodians:
Army - CR
Navy - OM
Air Force - 90

Preparing Activity:
DCA - DC
(Project IPSC-0166-02)

MIL-STD-1780
10 May 1984

Review Activities:

Army - SC, CR, AD

Navy - AS, YD, MC, OM, ND, NC, EC, SA

Air Force - 1, 11, 13, 17, 90, 99

Other Interest:

NSA - NS

TR1-TAC-TT

☆ U.S. GOVERNMENT PRINTING OFFICE: 1984-505-038/A4604

MIL-STD 1781

10 MAY 1984

MILITARY STANDARD

SIMPLE MAIL TRANSFER PROTOCOL



NO DELIVERABLE DATA

REQUIRED BY THIS DOCUMENT

IPSC/SLHC/TCTS

MIL-STD-1781
10 May 1984

DEPARTMENT OF DEFENSE
WASHINGTON, D.C. 20301

Simple Mail Transfer Protocol

MIL-STD-1781

1. This Military Standard is approved for use by all Departments and Agencies of the Department of Defense.
2. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to: Defense Communications Agency, ATTN: J110, 1860 Wiehle Avenue, Reston, Virginia 22090, by using the self-addressed Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document, or by letter.
3. Because of the rapid development in this standardization area, an alternative method of communication is offered. Forward responses using the MILNET to DCA-IAS @ DCA-EMS. Cooperation of the user is important to make this protocol meet Department of Defense needs.

MIL-STD-1781
10 May 1984

FOREWORD

This document specifies the Simple Mail Transfer Protocol (SMTP), a protocol designed to transfer mail reliably and efficiently. The document includes an introduction to SMTP with a model of operation, procedures, and specifications, including state diagrams.

MIL-STD-1781

10 May 1984

CONTENTS

		<u>Page</u>
Paragraph 1.	SCOPE - - - - -	1
1.1	Purpose - - - - -	1
1.2	Organization- - - - -	1
1.3	Application - - - - -	1
2.	REFERENCED DOCUMENTS- - - - -	2
2.1	Issues of documents - - - - -	2
2.2	Other publications- - - - -	2
3.	DEFINITIONS - - - - -	3
3.1	Definition of terms - - - - -	3
4.	THE SMTP MODEL- - - - -	5
4.1	SMTP design - - - - -	5
4.1.1	Mail command- - - - -	5
4.1.2	Transmission of mail- - - - -	5
4.1.2.1	Forward- and reverse-path - - - - -	5
4.1.2.2	Multiple recipients - - - - -	6
4.1.3	Mail syntax - - - - -	6
5.	THE SMTP PROCEDURES - - - - -	7
5.1	Introduction- - - - -	7
5.2	Mail- - - - -	7
5.2.1	MAIL command- - - - -	7
5.2.2	RCPT command- - - - -	7
5.2.3	DATA command- - - - -	7
5.2.4	Synopsis- - - - -	8
5.2.5	Example of the SMTP procedure - - - - -	8
5.3	Forwarding- - - - -	8
5.3.1	Example of forwarding - - - - -	9
5.4	Verifying and expanding - - - - -	9
5.4.1	Example of verifying a user name- - - - -	9
5.4.2	Example of expanding a mailing list - - - - -	10
5.4.3	Character string arguments- - - - -	10
5.5	Sending and mailing - - - - -	10
5.6	Opening and closing - - - - -	11
5.6.1	Example of connection opening - - - - -	11
5.6.2	Example of connection closing - - - - -	11
5.7	Relaying- - - - -	12
5.7.1	Message arrival - - - - -	12
5.7.2	First host identification - - - - -	12
5.7.3	Prevention of loops in error reporting- - - - -	12
5.7.4	Example of an undeliverable mail notification message- - - - -	13
5.8	Domains - - - - -	13
5.9	Changing roles- - - - -	13
5.9.1	Refusal to change roles - - - - -	13

MIL-STD-1781

10 May 1984

CONTENTS - Continued

	<u>Page</u>
Paragraph 6.	THE SMTP SPECIFICATIONS - - - - - 14
6.1	SMTP commands - - - - - 14
6.1.1	Command semantics - - - - - 14
6.1.1.1	HELLO (HELO)- - - - - 14
6.1.1.2	MAIL (MAIL) - - - - - 14
6.1.1.3	RECIPIENT (RCPT)- - - - - 14
6.1.1.4	DATA (DATA) - - - - - 15
6.1.1.4.1	Response and further action - - - - - 15
6.1.1.4.2	Example of return path and received timestamps- 16
6.1.1.5	SEND (SEND) - - - - - 16
6.1.1.6	SEND or MAIL (SOML) - - - - - 16
6.1.1.7	SEND and MAIL (SAML)- - - - - 17
6.1.1.8	RESET (RSET)- - - - - 17
6.1.1.9	VERIFY (VRFY) - - - - - 17
6.1.1.10	EXPAND (EXPN) - - - - - 17
6.1.1.11	HELP (HELP) - - - - - 17
6.1.1.12	NOOP (NOOP) - - - - - 17
6.1.1.13	QUIT (QUIT) - - - - - 18
6.1.1.14	TURN (TURN) - - - - - 18
6.1.2	Restrictions- - - - - 18
6.1.3	Command syntax- - - - - 18
6.1.3.1	List of SMTP commands - - - - - 19
6.1.3.2	SMTP syntax - - - - - 19
6.1.3.2.1	Special note- - - - - 21
6.1.3.3	Timestamp and return path lines - - - - - 21
6.1.3.3.1	Return path example - - - - - 22
6.1.3.3.2	Timestamp line example- - - - - 22
6.2	SMTP replies- - - - - 22
6.2.1	Reply codes by function groups- - - - - 23
6.2.2	Numeric order list of reply codes - - - - - 24
6.3	Sequencing of commands and replies- - - - - 25
6.3.1	Command-reply sequences - - - - - 25
6.4	State diagrams- - - - - 26
6.4.1	SMTP commands - - - - - 26
6.4.2	The DATA command- - - - - 27
6.5	Details - - - - - 27
6.5.1	Minimum implementation- - - - - 27
6.5.2	Transparency- - - - - 28
6.5.2.1	ASCII characters- - - - - 28
6.5.3	Sizes - - - - - 28
6.5.3.1	Error reply codes - - - - - 29
7.	SERVICES- - - - - 30
7.1	TCP transport service - - - - - 30
7.1.1	Connection establishment- - - - - 30
7.1.2	Data transfer - - - - - 30
7.2	X.25 transport service- - - - - 30

MIL-STD-1781

10 May 1984

CONTENTS - Continued

		<u>Page</u>
Paragraph 8.	THEORY OF REPLY CODES - - - - -	31
8.1	Introduction- - - - -	31
8.1.1	Values for the first digit- - - - -	31
8.1.1.1	Positive preliminary reply (1 yz) - - - - -	31
8.1.1.2	Positive completion reply (2 yz)- - - - -	31
8.1.1.3	Positive intermediate reply (3 yz)- - - - -	31
8.1.1.4	Transient negative completion reply (4 yz)- - -	31
8.1.1.5	Permanent negative completion reply (5 yz)- - -	31
8.1.2	Values for the second digit - - - - -	32
8.1.2.1	Syntax (x0z)- - - - -	32
8.1.2.2	Information (x1z) - - - - -	32
8.1.2.3	Connections (x2z) - - - - -	32
8.1.2.4	Unspecified (x3z) - - - - -	32
8.1.2.5	Unspecified (x4z) - - - - -	32
8.1.2.6	Mail system (x5z) - - - - -	32
8.1.3	Values for the third digit- - - - -	32
8.1.3.1	Special format- - - - -	32
9.	SCENARIOS - - - - -	33
9.1	Introduction- - - - -	33
9.2	A typical SMTP transaction scenario - - - - -	33
9.3	Aborted SMTP transaction scenario - - - - -	33
9.4	Relayed mail scenario - - - - -	34
9.4.1	Source host to relay host - - - - -	34
9.4.2	Relay host to destination host- - - - -	34
9.5	Verifying and sending scenario- - - - -	35
9.6	Sending and mailing scenarios - - - - -	35
9.6.1	Doing the sending and mailing scenario more efficiently- - - - -	36
9.7	Mailing list scenario - - - - -	36
9.7.1	Expanding the first list- - - - -	37
9.7.2	Expanding the second list - - - - -	37
9.7.3	Mailing to all via a relay host - - - - -	37
9.8	Forwarding scenarios- - - - -	38
9.9	Reading and delivering the mail - - - - -	38
9.9.1	Trying the mailbox at the first host- - - - -	38
9.9.2	Delivering the mail at the second host- - - - -	39
9.10	Too many recipients scenario- - - - -	39

MIL-STD-1781
10 May 1984

FIGURES

		<u>Page</u>
Figure 1	Model for SMTP use - - - - -	5
2	Representation of most of the SMTP commands - - -	27
3	The DATA command - - - - -	27

MIL-STD-1781
10 May 1984

1. SCOPE

1.1 Purpose. This standard establishes criteria for the Simple Mail Transfer Protocol (SMTP), a protocol designed to transfer mail reliably and efficiently.

1.2 Organization. This standard introduces the Simple Mail Transfer Protocol's design and procedures in sending and receiving mail and specifies the commands and other mechanisms needed to support SMTP. This standard also describes the uses of SMTP with various transport services.

1.2.1 Transport services. One of the most important features of SMTP is its capability to relay mail across transport service environments. A transport service provides an interprocess communication environment (IPCE). An IPCE may cover one network, several networks, or a subset of a network. It is important to realize that transport systems (or IPCEs) are not one-to-one with networks. A process can communicate directly with another process through any mutually known IPCE. Mail is an application or use of inter-process communication. Mail can be communicated between processes in different IPCEs by relaying through a process connected to two (or more) IPCEs. More specifically, mail can be relayed between hosts on different transport systems by a host on both transport systems.

1.3 Application. The Simple Mail Transfer Protocol is approved for use in all DoD packet switching networks which connect or have the potential for utilizing connectivity across network and subnetwork boundaries and which require a mail transfer service. The term network as used herein includes Local Area Networks.

10 May 1984

2. REFERENCED DOCUMENTS

2.1 Issues of documents. The following documents of the issue in effect on date of invitation for bids or request for proposal, form a part of this standard to the extent specified herein.

STANDARDS

FEDERAL

FED-STD-1037

Glossary of Telecommunication Terms

MILITARY

MIL-STD-1778

Transmission Control Protocol

2.2 Other publications. The following documents form a part of this standard to the extent specified herein. Unless otherwise indicated, the issue in effect on date of invitation for bids or request for proposal shall apply. (The provisions of this paragraph are under consideration.)

MIL-STD-1781

10 May 1984

3. DEFINITIONS

3.1 Definition of terms. The definition of terms used in this standard shall comply with FED-STD-1037. Terms and definitions unique to MIL-STD-1781 are contained herein.

- a. Command
A request for a mail service action sent by the sender-SMTP to the receiver-SMTP.
- b. Domain
The hierarchially structured global character string address of a host computer in the mail system.
- c. End of mail data indication
A special sequence of characters that indicates the end of the mail data. In particular, the five characters carriage return, line feed, period, carriage return, line feed, in that order.
- d. Host
A computer in the internetwork environment on which mailboxes or SMTP processes reside.
- e. Line
A sequence of ASCII characters ending with a <CR>.
- f. Mail data
A sequence of ASCII characters of arbitrary length, which conforms to the standard set in the Standard for the Format of ARPA Internet Text Messages.
- g. Mailbox
A character string (address) which identifies a user to whom mail is to be sent. Mailbox normally consists of the host and user specifications. The standard mailbox naming convention is defined to be "user@domain". Additionally, the "container" in which mail is stored.
- h. Receiver-SMTP process
A process which transfers mail in cooperation with a sender-SMTP process. It waits for a connection to be established via the transport service. It receives SMTP commands from the sender-SMTP, sends replies, and performs the specified operations.
- i. Reply
A reply is an acknowledgment (positive or negative) sent from receiver to sender via the transmission channel in response to a command. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

MIL-STD-1781

10 May 1984

- j. Sender-SMTP process
A process which transfers mail in cooperation with a receiver-SMTP process. A local language may be used in the user interface command/reply dialogue. The sender-SMTP initiates the transport service connection. It initiates SMTP commands, receives replies, and governs the transfer of mail.
- k. Session
The set of exchanges that occur while the transmission channel is open.
- l. Transaction
The set of exchanges required for one message to be transmitted for one or more recipients.
- m. Transmission channel
A full-duplex communication path between a sender-SMTP and a receiver-SMTP for the exchange of commands, replies, and mail text.
- n. Transport service
Any reliable stream-oriented data communication services. For example, NCP, TCP, NITS.
- o. <CRLF>
The characters carriage return and line feed (in that order).
- p. <SP>
The space character.

MIL-STD-1781

10 May 1984

4. THE SMTP MODEL

4.1 SMTP design. The SMTP design is based on the following model of communication: as the result of a user mail request, the sender-SMTP establishes a two-way transmission channel to a receiver-SMTP. The receiver-SMTP commands are generated by the sender-SMTP and sent to the receiver-SMTP. SMTP replies are sent from the receiver-SMTP to the sender-SMTP in response to the commands.

4.1.1 Mail command. Once the transmission channel is established, the SMTP-sender sends a MAIL command indicating the sender of the mail. If the SMTP-receiver can accept mail it responds with an OK reply. The SMTP-sender then sends a RCPT command identifying a recipient of the mail. If the SMTP-receiver can accept mail for that recipient it responds with an OK reply; if not, it responds with a reply rejecting that recipient (but not the whole mail transaction). The SMTP-sender and SMTP-receiver may negotiate several recipients. When the recipients have been negotiated the SMTP-sender sends the mail data, terminating with a special sequence. If the SMTP-receiver successfully processes the mail data it responds with an OK reply. The dialog is purposely lock-step, one-at-a-time.



FIGURE 1. Model for SMTP use.

4.1.2 Transmission of mail. The SMTP provides mechanisms for the transmission of mail; directly from the sending user's host to the receiving user's host when the two hosts are connected to the same transport service, or via one or more relay SMTP-servers when the source and destination hosts are not connected to the same transport service. To be able to provide the relay capability, the SMTP-server must be supplied with the name of the ultimate destination host as well as the destination mailbox name.

4.1.2.1 Forward- and reverse-path. The argument to the MAIL command is a reverse-path, which specifies who the mail is from. The argument to the RCPT command is a forward-path, which specifies who the mail is to. The forward-path is a source route, while the reverse-path is a return route (which may be used to return a message to the sender when an error occurs with a relayed message).

MIL-STD-1781

10 May 1984

4.1.2.2 Multiple recipients. When the same message is sent to multiple recipients the SMTP encourages the transmission of only one copy of the data for all the recipients at the same destination host.

4.1.3 Mail syntax. The mail commands and replies have a rigid syntax. Replies also have a numeric code. The following examples use actual commands and replies. The complete lists of commands and replies appears in Section 4 on specifications. Commands and replies are not case sensitive. That is, a command or reply word may be upper case, lower case, or any mixture of upper and lower case. Note that this is not true of mailbox user names. For some hosts the user name is case sensitive, and SMTP implementations must take case to preserve the case of user names as they appear in mailbox arguments. Host names are not case sensitive. Commands and replies are composed of characters from the ASCII character set. When the transport service provides an 8-bit byte (octet) transmission channel, each 7-bit character is transmitted right justified in an octet with the high order bit cleared to zero. When specifying the general form of a command or reply, an argument (or special symbol) will be denoted by a meta-linguistic variable (or constant), for example, "<string>" or "<reverse-path>". Here the angle brackets indicate these are meta-linguistic variables. However, some arguments use the angle brackets literally. For example, an actual reverse-path is enclosed in angle brackets, i.e., "<John.Smith@USC-ISI.ARPA>" is an instance of <reverse-path> (the angle brackets are actually transmitted in the command or reply).

MIL-STD-1781

10 May 1984

5. THE SMTP PROCEDURES

5.1 Introduction. This section presents the procedures used in SMTP in several parts. First comes the basic mail procedure defined as a mail transaction. Following this are descriptions of forwarding mail, verifying mailbox names and expanding mailing lists, sending to terminals instead of or in combination with mailboxes, and the opening and closing exchanges. At the end of this section are comments on relaying, a note on mail domains, and a discussion of changing roles. Throughout this section are examples of partial command and reply sequences.

5.2 Mail. There are three steps to SMTP mail transactions. The transaction is started with a MAIL command which gives the sender identification. A series of one or more RCPT commands follows giving the receiver information. Then a DATA command gives the mail data. And finally, the end of mail data indicator confirms the transaction.

5.2.1 MAIL command. The first step in the procedure is the MAIL command. The <reverse-path> contains the source mailbox.

MAIL <SP> FROM:<reverse-path> <CRLF>

This command tells the SMTP-receiver that a new mail transaction is starting and to reset all its state tables and buffers, including any recipients or mail data. It gives the reverse-path which can be used to report errors. If accepted, the receiver-SMTP returns a 250 OK reply. The <reverse-path> can contain more than just a mailbox. The <reverse-path> is a reverse source routing list of hosts and source mailbox. The first host in the <reverse-path> should be the host sending this command.

5.2.2 RCPT command. The second step in the procedure is the RCPT command.

RCPT <SP> TO:<forward-path> <CRLF>

This command gives a forward-path identifying one recipient. If accepted, the receiver-SMTP returns a 250 OK reply, and stores the forward-path. If the recipient is unknown the receiver-SMTP returns a 550 Failure reply. This second step of the procedure can be repeated any number of times. The <forward-path> can contain more than just a mailbox. The <forward-path> is a source routing list of hosts and the destination mailbox. The first host in the <forward-path> should be the host receiving this command.

5.2.3 DATA command. The third step in the procedure is the DATA command.

DATA <CRLF>

If accepted, the receiver-SMTP returns a 354 Intermediate reply and considers all succeeding lines to be the message text. When the end of text is received and stored the SMTP-receiver sends a 250 OK reply. Since the mail data is sent on the transmission channel the end of the mail data must be indicated

MIL-STD-1781

10 May 1984

so that the command and reply dialog can be resumed. SMTP indicates the end of the mail data by sending a line containing only a period. A transparency procedure is used to prevent this from interfering with the user's test (see Section 4.5.2). The end of mail data indicator also confirms the mail transaction and tells the receiver-SMTP to now process the stored recipients and mail data. If accepted, the receiver-SMTP returns a 250 OK reply. The DATA command should fail only if the mail transaction was incomplete (for example, no recipients), or if resources are not available.

5.2.4 Synopsis. The above procedure is an example of a mail transaction. These commands must be used only in the order discussed above. Paragraph 5.2.5 illustrates the use of these commands in a mail transaction.

5.2.5 Example of the SMTP procedure. This SMTP example shows mail sent by Smith at host Alpha.ARPA, to Jones, Green, and Brown at host Beta.ARPA. Here we assume that host Alpha contacts host Beta directly.

```
S: MAIL FROM:<Smith@Alpha.ARPA>
R: 250 OK

S: RCPT TO:<Jones@Beta.ARPA>
R: 250 OK

S: RCPT TO:<Green@Beta.ARPA>
R: 550 No such user here

S: RCPT TO:<Brown@Beta.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

The mail has now been accepted for Jones and Brown. Green did not have a mailbox at host Beta.

5.3 Forwarding. There are some cases where the destination information in the <forward-path> is incorrect, but the receiver-SMTP knows the correct destination. In such cases, one of the following replies should be used to allow the sender to contact the correct destination.

- a. 251 User not local; will forward to <forward-path>. This reply indicates that the receiver-SMTP knows the user's mailbox is on another host and indicates the correct forward-path to use in the future. Note that either the host or user or both may be different. The receiver takes responsibility for delivering the message.

MIL-STD-1781

10 May 1984

- b. 551 User not local; please try <forward-path>. This reply indicates that the receiver-SMTP knows the user's mailbox is on another host and indicates the correct forward-path to use. Note that either the host or user or both may be different. The receiver refuses to accept mail for this user, and the sender must either redirect the mail according to the information provided or return an error response to the originating user.

Paragraph 5.3.1 illustrates the use of these responses.

5.3.1 Example of forwarding. Either:

S: RCPT TO:<Postel@USC-ISI.ARPA>
R: 251 User not local; will forward to <Postel@USC-ISIF.ARPA>

Or:

S: RCPT TO:<Paul@USC-ISIB.ARPA>
R: 551 User not local; please try <Mockapetris@USC-ISIF.ARPA>

5.4 Verifying and expanding. SMTP provides as additional features, commands to verify a user name or expand a mailing list. This is done with the VRFY and EXPN commands, which have character string arguments. For the VRFY command, the string is a user name, and the response may include the full name of the user and must include the mailbox of the user. For the EXPN command, the string identifies a mailing list, and the multiline response may include the full name of the users and must give the mailboxes on the mailing list. "User name" is a fuzzy term and used purposely. If a host implements the VRFY or EXPN commands then at least local mailboxes must be recognized as "user names". If a host chooses to recognize other strings as "user names" that is allowed. In some hosts the distinction between a mailing list and an alias for a single mailbox is a bit fuzzy, since a common data structure may hold both types of entries, and it is possible to have mailing lists of one mailbox. If a request is made to verify a mailing list, a positive response can be given if on receipt of a message so addressed it will be delivered to everyone on the list, otherwise an error should be reported (e.g., "550 That is a mailing list, not a user"). If a request is made to expand a user name a positive response can be formed by returning a list containing one name, or an error can be reported (e.g., "550 That is a user name, not a mailing list"). In the case of a multiline reply (normal for EXPN) exactly one mailbox is to be specified on each line of the reply. In the case of an ambiguous request, for example, "VRFY Smith", where there are two Smith's the response must be "553 User ambiguous". The case of verifying a user name is straightforward as shown in paragraph 5.4.1.

5.4.1 Example of verifying a user name. Either:

S: VRFY Smith
R: 250 Fred Smith <Smith@USC-ISIF.ARPA>

MIL-STD-1781

10 May 1984

Or:

S: VRFY Smith

R: 251 User not local; will forward to <Smith@USC-ISIQ.ARPA>

Or:

S: VRFY Jones

R: 550 String does not match anything.

Or:

S: VRFY Jones

R: 551 User not local; please try <Jones@USC-ISIQ.ARPA>

Or:

S: VRFY Gourzenkyinplatz

R: 553 User ambiguous.

5.4.2 Example of expanding a mailing list. The case of expanding a mailbox list requires a multiline reply as either:

S: EXPN Example-People

R: 250-Jon Postal <Postal@USC-ISIF.ARPA>

R: 250-Fred Fonebone <Fonebone@USC-ISIQ.ARPA>

R: 250-Sam Q. Smith <SQSmith@USC-ISIQ.ARPA>

R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>

R: 250-<joe@foo-unix.ARPA>

R: 250 <xyz@bar-unix.ARPA>

Or:

S: EXPN Executive-Washroom-List

R: 550 Access Denied to You.

5.4.3 Character string arguments. The character string arguments of the VRFY and EXPN commands cannot be further restricted due to the variety of implementations of the user name and mailbox list concepts. On some systems it may be appropriate for the argument of the EXPN command to be a file name for a file containing a mailing list, but again there is a variety of file naming conventions in the Internet. The VRFY and EXPN commands are not included in the minimum implementation (paragraph 6.5.1) and are not required to work across relays when they are implemented.

5.5 Sending and mailing. The main purpose of SMTP is to deliver messages to user's mailboxes. A very similar service provided by some hosts is to deliver messages to user's terminals (provided the user is active on the host). The delivery to the user's mailbox is called "mailing," the delivery to the user's terminal is called "sending." Because in many hosts the implementation of sending is nearly identical to the implementation of mailing

MIL-STD-1781

10 May 1984

5.6.2 Example of connection closing.

S: QUIT

R: 221 BBN-UNIX.ARPA Service closing transmission channel

5.7 Relaying. The forward-path may be a source route of the form "@ONE,@TWO:JOE@THREE", where ONE, TWO, and THREE are hosts. This form is used to emphasize the distinction between an address and a route. The mailbox is an absolute address, and the route is information about how to get there. The two concepts should not be confused. Conceptually, the elements of the forward-path are moved to the reverse-path as the message is relayed from one server-SMTP to another. The reverse-path is a reverse source route, (i.e., a source route from the current location of the message to the originator of the message). When a server-SMTP deletes its identifier from the forward-path and inserts it into the reverse-path, it must use the name it is known by in the environment it is sending into, not the environment the mail came from, in case the server-SMTP is known by different names in different environments.

5.7.1 Message arrival. If when the message arrives at an SMTP the first element of the forward-path is not the identifier of that SMTP the element is not deleted from the forward-path and is used to determine the next SMTP to send the message to. In any case, the SMTP adds its own identifier to the reverse-path. Using source routing the receiver-SMTP receives mail to be relayed to another server-SMTP. The receiver-SMTP may accept or reject the task of relaying the mail in the same way it accepts or rejects mail for a local user. The receiver-SMTP transforms the command arguments by moving its own identifier from the forward-path to the beginning of the reverse-path. The receiver-SMTP then becomes a sender-SMTP, establishes a transmission channel to the next SMTP in the forward-path, and sends it the mail.

5.7.2 First host identification. The first host in the reverse-path should be the host sending the SMTP commands, and the first host in the forward-path should be the host receiving the SMTP commands. Notice that the forward-path and reverse-path appear in the SMTP commands and replies, but not necessarily in the message. That is, there is no need for these paths and especially this syntax to appear in the "To:", "From:", "CC:", etc., fields of the message header. If a server-SMTP has accepted the task of relaying the mail and later finds that the forward-path is incorrect or that the mail cannot be delivered for whatever reason, then it must construct an "undeliverable mail" notification message and send it to the originator of the undeliverable mail (as indicated by the reverse-path).

5.7.3 Prevention of loops in error reporting. This notification message must be from the server-SMTP at this host. Of course, server-SMTPs should not send notification messages about problems with notification messages. One way to prevent loops in error reporting is to specify a null reverse-path in the MAIL command of a notification message. When such a message is relayed it is permissible to leave the reverse-path null. A MAIL command with a null reverse-path appears as MAIL FROM:<>. An undeliverable mail

MIL-STD-1781
10 May 1984

these two functions are combined in SMTP. However the sending commands are not included in the required minimum implementation (paragraph 6.5.1). Users should have the ability to control the writing of messages on their terminals. Most hosts permit the users to accept or refuse such messages. The following three commands are defined to support the sending options. These are used in the mail transaction instead of the MAIL command and inform the receiver-SMTP of the special semantics of this transaction:

- a. SEND <SP> FROM:<reverse-path> <CRLF>. The SEND command requires that the mail data be delivered to the user's terminal. If the user is not active (or not accepting terminal messages) on the host a 450 reply may return to a RCPT command. The mail transaction is successful if the message is delivered to the terminal. The same reply code which is used for the MAIL commands is used for this command.
- b. SOML <SP> FROM:<reverse-path> <CRLF>. The SEND or MAIL command requires that the mail data be delivered to the user's terminal if the user is active (and accepting terminal messages) on the host. If the user is not active (or not accepting terminal messages) then the mail data is entered into the user's mailbox. The mail transaction is successful if the message is delivered either to the terminal or the mailbox. The same reply code which is used for the MAIL commands is used for this command.
- c. SANL <SP> FROM:<reverse-path> <CRLF>. The SEND and MAIL command requires that the mail data be delivered to the user's terminal if the user is active (and accepting terminal messages) on the host. In any case the mail data is entered into the user's mailbox. The mail transaction is successful if the message is delivered to the mailbox. The same reply code which is used for the MAIL commands is used for this command.

5.6 Opening and closing. At the time the transmission channel is opened there is an exchange to ensure that the hosts are communicating with the hosts they think they are. The following two commands are used in transmission channel opening and closing:

- a. HELO <SP> <domain> <CRLF>
- b. QUIT <CRLF>

In the HELO command the host sending the command identifies itself; the command may be interpreted as saying "Hello, I am <domain>".

5.6.1 Example of connection opening.

```
R: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIP.ARPA
R: 250 BBN-UNIX.ARPA
```

MIL-STD-1781

10 May 1984

notification message is shown in paragraph 5.7.4. This notification is in response to a message originated by JOE at HOSTW and sent via HOSTX to HOSTY with instructions to relay it on to HOSTZ. What we see in the example is the transaction between HOSTY and HOSTX, which is the first step in the return of the notification message.

5.7.4 Example of an undeliverable mail notification message.

```
S: MAIL FROM:<>
R: 250 ok
S: RCPT TO:<@HOSTX.ARPA:JOE@HOSTW.ARPA>
R: 250 ok
S: DATA
R: 354 send the mail data, end with .
S: Date: 23 Oct 81 11:22:33
S: From: SMTP@HOSTY.ARPA
S: To: JOE@HOSTW.ARPA
S: Subject: Mail System Problem
S:
S: Sorry JOE, your message to SAM@HOSTZ.ARPA lost.
S: HOSTZ.ARPA said this:
S: "550 No Such User"
S: .
R: 250 ok
```

5.8 Domains. Domains are a recently introduced concept in the ARPA Internet mail system. The use of domains changes the address space from a flat global space of simple character string host names to a hierarchically structured rooted tree of global addresses. The host name is replaced by a domain and host designator which is a sequence of domain element strings separated by periods with the understanding that the domain elements are ordered from the most specific to the most general. For example, "USC-ISIF.ARPA", "Fred.Cambridge.UK", and "PC7.LCS.MIT.ARPA" might be host-and-domain identifiers. Whenever domain names are used in SMTP only the official names are used, the use of nicknames or aliases are not allowed.

5.9 Changing roles. The TURN command may be used to reverse the roles of the two programs communicating over the transmission channel. If program-A is currently the sender-SMTP and it sends the TURN command and receives an ok reply (250) then program-A becomes the receiver-SMTP. If program-B is currently the receiver-SMTP and it receives the TURN command and sends an ok reply (250) then program-B becomes the sender-SMTP.

5.9.1 Refusal to change roles. To refuse to change roles the receiver sends the 502 reply. Please note that this command is optional. It would not normally be used in situations where the transmission channel is TCP. However, when the cost of establishing the transmission channel is high, this command may be quite useful. For example, this command may be useful in supporting the mail exchange using the public switched telephone system as a transmission channel, especially if some hosts poll other hosts for mail exchanges.

MIL-STD-1781
10 May 1984

6. THE SMTP SPECIFICATIONS

6.1 SMTP commands.

6.1.1 Command semantics. The SMTP commands define the mail transfer or the mail system function requested by the user. SMTP commands are character strings terminated by <CRLF>. The command codes themselves are alphabetic characters terminated by <SP> if parameters follow and <CRLF> otherwise. The syntax of mailboxes must conform to receiver site conventions. The SMTP commands are discussed below. The SMTP replies are discussed in the Section 4.2. A mail transaction involves several data objects which are communicated as arguments to different commands. The reverse-path is the argument of the MAIL command, the forward-path is the argument of the RCPT command, and the mail data is the argument of the DATA command. These arguments or data objects must be transmitted and held pending the confirmation communicated by the end of mail data indication which finalizes the transaction. The model for this is that distinct buffers are provided to hold the types of data objects, that is, there is a reverse-path buffer, a forward-path buffer, and a mail data buffer. Specific commands cause information to be appended to a specific buffer, or cause one or more buffers to be cleared.

6.1.1.1 HELLO (HELO). This command is used to identify the sender-SMTP to the receiver-SMTP. The argument field contains the host name of the sender-SMTP. The receiver-SMTP identifies itself to the sender-SMTP in the connection greeting reply, and in the response to this command. This command and an OK reply to it confirm that both the sender-SMTP and the receiver-SMTP are in the initial state, that is, there is no transaction in progress and all state tables and buffers are cleared.

6.1.1.2 MAIL (MAIL). This command is used to initiate a mail transaction in which the mail data is delivered to one or more mailboxes. The argument field contains a reverse-path. The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different). In some types of error reporting messages (for example, undeliverable mail notifications) the reverse-path may be null (see paragraph 5.7.4). This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

6.1.1.3 RECIPIENT (RCPT). This command is used to identify an individual recipient of the mail data; multiple recipients are specified by multiple use of this command. The forward-path consists of an optional list of hosts and a required destination mailbox. When the list of hosts is present, it is a source route and indicates that the mail must be relayed to the next host on the list. If the receiver-SMTP does not implement the relay function

MIL-STD-1781
10 May 1984

it may use the same reply it would for an unknown local user (550). When mail is relayed, the relay host must remove itself from the beginning forward-path and put itself at the beginning of the reverse-path. When mail reaches its ultimate destination (the forward-path contains only a destination mail-box), the receiver-SMTP inserts it into the destination mailbox in accordance with its host mail conventions. For example, mail received at relay host A with arguments

```
FROM:<USERX@HGSTY.ARPA>  
TO:<@HOSTA.ARPA,@HOSTB.ARPA:USERC@HOSTD.ARPA>  
will be relayed on to host B with arguments
```

```
FROM:<@HOSTA.ARPA:USERX@HOSTY.ARPA>  
TO:<@HOSTB.ARPA:USERC@HOSTD.ARPA>.
```

This command causes its forward-path argument to be appended to the forward-path buffer.

6.1.1.4 DATA (DATA). The receiver treats the lines following the command as mail data from the sender. This command causes the mail data from this command to be appended to the mail data buffer. The mail data may contain any of the 128 ASCII character codes. The mail data is terminated by a line containing only a period, that is the character sequence "<CRLF>.<CRLF>" (see paragraph 6.5.2 on Transparency). This is the end of mail data indication. The end of mail data indication requires that the receiver must now process the stored mail transaction information. This processing consumes the information in the reverse-path buffer, the forward-path buffer, and the mail data buffer, and on the completion of this command these buffers are cleared. If the processing is successful the receiver must send an OK reply. If the processing fails completely the receiver must send a failure reply. When the receiver-SMTP accepts a message either for relaying or for final delivery it inserts at the beginning of the mail data a timestamp line. The timestamp line indicates the identity of the host that sent the message, and the identity of the host that received the message (and is inserting this timestamp), and the date and time the message was received. Relayed messages will have multiple timestamp lines. When the receiver-SMTP makes the "final delivery" of a message it inserts at the beginning of the mail data a return path line. The return path line preserves the information in the <reverse-path> from the MAIL command. Here, final delivery means the message leaves the SMTP world. Normally, this would mean it has been delivered to the destination user, but in some cases it may be further processed and transmitted by another mail system. It is possible for the mailbox in the return path to be different from the actual sender's mailbox; for example, if error responses are to be delivered to a special error handling mailbox rather than to the message senders. The preceding implies that the final mail data will begin with a return path line, followed by one or more time stamp lines. These lines will be followed by the mail data header and body. See paragraph 8.

6.1.1.4.1 Response and further action. Special mention is needed of the response and further action required when the processing following the end of mail data indication is partially successful. This could arise if, after

MIL-STD-1781

10 May 1984

accepting several recipients and the mail data, the receiver-SMTP finds that the mail data can be successfully delivered to some of the recipients, but it cannot be to others (for example, due to mailbox space allocation problems). In such a situation, the response to the DATA command must be an OK reply. But, the receiver-SMTP must compose and send an "undeliverable mail" notification message to the originator of the message. Either a single notification which lists all of the recipients that failed to get the message, or separate notification messages must be sent for each failed recipient (see paragraph 7). All undeliverable mail notification messages are sent using the MAIL command (even if they result from processing a SEND, SOHL, or SAML command).

6.1.1.4.2 Example of return path and received timestamps.

```
Return-Path: <GHI.ARPA,DEF.ARPA,ABC.ARPA:JOE@ABC.ARPA>  
Received: from GHI.ARPA by JKL.ARPA ; 27 Oct 81 15:27:39 PST  
Received: from DEF.ARPA by GHI.ARPA ; 27 Oct 81 15:15:13 PST  
Received: from ABC.ARPA by DEF.ARPA ; 27 Oct 81 15:01:59 PST  
Date: 27 Oct 81 15:01:01 PST  
From: JOE@ABC.ARPA  
Subject: Improved Mailing System Installed  
To: SAM@JKL.ARPA  
(Message).
```

6.1.1.5 SEND (SEND). This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals. The argument field contains a reverse-path. This command is successful if the message is delivered to a terminal. The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different). This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

6.1.1.6 SEND or MAIL (SOHL). This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals or mailboxes. For each recipient the mail data is delivered to the recipient's terminal if the recipient is active on the host (and accepting terminal messages), otherwise to the recipient's mailbox. The argument field contains a reverse-path. This command is successful if the message is delivered to a terminal or the mailbox. The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different). This

MIL-STD-1781

10 May 1984

command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

6.1.1.7 SEND and MAIL (SAML). This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals and mailboxes. For each recipient the mail data is delivered to the recipient's terminal if the recipient is active on the host (and accepting terminal messages), and for all recipients to the recipient's mailbox. The argument field contains a reverse-path. This command is successful if the message is delivered to the mailbox. The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different). This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

6.1.1.8 RESET (RSET). This command specifies that the current mail transaction is to be aborted. Any stored sender, recipients, and mail data must be discarded, and all buffers and state tables cleared. The receiver must send an OK reply.

6.1.1.9 VERIFY (VRFY). This command asks the receiver to confirm that the argument identifies a user. If it is a user name, the full name of the user (if known) and the fully specified mailbox are returned. This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

6.1.1.10 EXPAND (EXPN). This command asks the receiver to confirm that the argument identifies a mailing list, and if so, to return the membership of that list. The full name of the users (if known) and the fully specified mailboxes are returned in a multiline reply. This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

6.1.1.11 HELP (HELP). This command causes the receiver to send helpful information to the sender of the HELP command. The command may take an argument (e.g., any command name) and return more specific information as a response. This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

6.1.1.12 NOOP (NOOP). This command does not affect any parameters or previously entered commands. It specifies no action other than that the receiver send an OK reply. This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

MIL-STD-1781

10 May 1984

6.1.1.13 QUIT (QUIT). This command specifies that the receiver must send an OK reply, and then close the transmission channel. The receiver should not close the transmission channel until it receives and replies to a QUIT command (even if there was an error). The sender should not close the transmission channel until it send a QUIT command and receives the reply (even if there was an error response to a previous command). If the connection is closed prematurely the receiver should act as if a RSET command had been received (canceling any pending transaction, but not undoing any previously completed transaction), the sender should act as if the command or transaction in progress had received a temporary error (4xx).

6.1.1.14 TURN (TURN). This command specifies that the receiver must either (1) send an OK reply and then take on the role of the sender-SMTP, or (2) send a refusal reply and retain the role of the receiver-SMTP. If program-A is currently the sender-SMTP and it sends the TURN command and receives an OK reply (250) then program-A becomes the receiver-SMTP. Program-A is then in the initial state as if the transmission channel just opened, and it then sends the 220 service ready greeting. If program-B is currently the receiver-SMTP and it receives the TURN command and sends an OK reply (250) then program-B becomes the sender-SMTP. Program-B is then in the initial state as if the transmission channel just opened, and it then expects to receive the 220 service ready greeting. To refuse to change roles the receiver sends the 502 reply.

6.1.2 Restrictions. There are restrictions on the order in which these commands may be used. The first command in a session must be the HELO command. The HELO command may be used later in a session as well. If the HELO command argument is not acceptable a 501 failure reply must be returned and the receiver-SMTP must stay in the same state. The NOOP, HELP, EXPN, and VRFY commands can be used at any time during a session. The MAIL, SEND, SOHL, or SAML commands begin a mail transaction. Once started a mail transaction consists of one of the transaction beginning commands, one or more RCPT commands, and a DATA command, in that order. A mail transaction may be aborted by the RSET command. There may be zero or more transactions in a session. If the transaction beginning command argument is not acceptable a 501 failure reply must be returned and the receiver-SMTP must stay in the same state. If the commands in a transaction are out of order a 503 failure reply must be returned and the receiver-SMTP must stay in the same state. The last command in a session must be the QUIT command. The QUIT command can not be used at any other time in a session.

6.1.3 Command syntax. The commands consist of a command code followed by an argument field. Command codes are four alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus, any of the following may represent the mail command:

MAIL Mail mail Mail mAIl

This also applies to any symbols representing parameter values, such as "TO" or "to" for the forward-path. Command codes and the argument fields are separated by one or more spaces. However, within the reverse-path and forward-path arguments case is important. In particular, in some hosts the user

MIL-STD-1781
10 May 1984

"smith" is different from the user "Smith." The argument field consists of a variable length character string ending with the character sequence <CRLF>. The receiver is to take no action until this sequence is received. Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

6.1.3.1 List of SMTP commands. The following are the SMTP commands:

- a. HELO <SP> <domain> <CRLF>
- b. MAIL <SP> FROM:<reverse-path> <CRLF>
- c. RCPT <SP> TO:<forward-path> <CRLF>
- d. DATA <CRLF>
- e. RSET <CRLF>
- f. SEND <SP> FROM:<reverse-path> <CRLF>
- g. SOML <SP> FROM:<reverse-path> <CRLF>
- h. SAML <SP> FROM:<reverse-path> <CRLF>
- i. VRFY <SP> <string> <CRLF>
- j. EXPN <SP> <string> <CRLF>
- k. HELP [<SP> <string>] <CRLF>
- l. NOOP <CRLF>
- m. QUIT <CRLF>
- n. TURN <CRLF>

6.1.3.2 SMTP syntax. The syntax of the above argument fields (using BNF notation where applicable) is given below. The "... " notation indicates that a field may be repeated one or more times.

- a. <reverse-path> ::= <path>
- b. <forward-path> ::= <path>
- c. <path> ::= "<" [<a-d-l> ":"] <mailbox> ">"
- d. <a-d-l> ::= <at-domain> | <at-domain> ",", <a-d-l>
- e. <at-domain> ::= "@" <domain>
- f. <domain> ::= <element> | <element> "." <domain>

MIL-STD-1781

10 May 1984

- g. <element> ::= <name> | "#" <number> | "[" <dotnum> "]"
- h. <mailbox> ::= <local-part> "@" <domain>
- i. <local-part> ::= <dot-string> | <quoted-string>
- j. <name> ::= <a> <ldh-str> <let-dig>
- k. <ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>
- l. <let-dig> ::= <a> | <d>
- m. <let-dig-hyp> ::= <a> | <d> | "-"
- n. <dot-string> ::= <string> | <string> "." <dot-string>
- o. <string> ::= <char> | <char> <string>
- p. <quoted-string> ::= "" <qtext> ""
- q. <qtext> ::= \ <x> | \ <x> <qtext> | <q> | <q> <qtext>
- r. <char> ::= <c> | \ <x>
- s. <dot.num> ::= <enum> "." <enum> "." <enum> "." <enum>
- t. <number> ::= <d> | <d> <number>
- u. <CRLF> ::= <CR> <LF>
- v. <CR> ::= the carriage return character (ASCII code 13)
- w. <LF> ::= the line feed character (ASCII code 10)
- x. <SP> ::= the space character (ASCII code 32)
- y. <enum> ::= one, two, or three digits representing a decimal integer value in the range 0 through 255
- z. <a> ::= any one of the 52 alphabetic characters A through Z in upper case and a through z in lower case
- aa. <c> ::= any one of the 128 ASCII characters, but not any <special> or <SP>
- bb. <d> ::= any one of the ten digits 0 through 9
- cc. <q> ::= any one of the 128 ASCII characters except <CR>, <LF>, quote ("), or backslash (\)
- dd. <x> ::= any one of the 128 ASCII characters (no exceptions)

MIL-STD-1781

10 May 1984

```

ee. <special> ::= "<" | ">" | "(" | ")" | "[" | "]" | "\" | "."
                | "," | ";" | ":" | "e" | " " | the control
                characters (ASCII codes 0 through 31 inclusive and
                127)

```

6.1.3.2.1 Special note. Note that the backslash, "\", is a quote character, which is used to indicate that the next character is to be used literally (instead of its normal interpretation). For example, "Joe,Smith" could be used to indicate a single nine character user field with comma being the fourth character of the field. Hosts are generally known by names which are translated to addresses in each host. Note that the name elements of domains are the official names -- no use of nicknames or aliases is allowed. Sometimes a host is not known to the translation function and communication is blocked. To bypass this barrier two numeric forms are also allowed for host "names". One form is a decimal integer prefixed by a pound sign, "#", which indicates the number is the address of the host. Another form is four small decimal integers separated by dots and enclosed by brackets, e.g., "[123.255.37.2]", which indicates a 32-bit ARPA Internet Address in four 8-bit fields.

6.1.3.3 Timestamp and return path lines. The timestamp line and the return path line are formally defined as follows:

- a. `<return-path-line> ::= "Return-Path:" <SP><reverse-path><CRLF>`
- b. `<time-stamp-line> ::= "Received:" <SP> <stamp> <CRLF>`
- c. `<stamp> ::= <from-domain> <by-domain> <opt-info> ";"
 <daytime>`
- d. `<from-domain> ::= "FROM" <SP> <domain> <SP>`
- e. `<by-domain> ::= "BY" <SP> <domain> <SP>`
- f. `<opt-info> ::= [<via>] [<with>] [<id>] [<for>]`
- g. `<via> ::= "VIA" <SP> <link> <SP>`
- h. `<with> ::= "WITH" <SP> <protocol> <SP>`
- i. `<id> ::= "ID" <SP> <string> <SP>`
- j. `<for> ::= "FOR" <SP> <path> <SP>`
- k. `<link> ::=` The standard names for links are registered with
 the Network Information Center.
- l. `<protocol> ::=` The standard names for protocols are
 registered with the Network Information Center.
- m. `<daytime> ::= <SP> <date> <SP> <time>`

MIL-STD-1781
10 May 1984

- n. <date> ::= <dd> <SP> <mon> <SP> <yy>
- o. <time> ::= <hh> ":" <mm> ":" <ss> <SP> <zone>
- p. <dd> ::= the one or two decimal integer day of the month in the range 1 to 31.
- q. <mon> ::= "JAN" ! "FEB" ! "MAR" ! "APR" ! "MAY" ! "JUN"
"JUL" ! "AUG" ! "SEP" ! "OCT" ! "NOV" ! "DEC"
- r. <yy> ::= the two decimal integer year of the century in the range 00 to 99.
- s. <hh> ::= the two decimal integer hour of the day in the range 00 to 24.
- t. <mm> ::= the two decimal integer minute of the hour in the range 00 to 59.
- u. <ss> ::= the two decimal integer second of the minute in the range 00 to 59.
- v. <zone> ::= "UT" for Universal Time (the default) or other time zone designator.

6.1.3.3.1 Return path example.

Return-Path: <@CHARLIE.ARPA,@BAKER.ARPA:JOE@ABLE.ARPA>

6.1.3.3.2 Timestamp line example.

Received: FROM ABC.ARPA BY XYZ.ARPA ; 22 OCT 81 09:23:59 PDT

Received: from ABC.ARPA by XYZ.ARPA via TELENET with X25
id M12345 for Smith@PDQ.ARPA ; 22 OCT 81 09:23:59 PDT

6.2 SMTP replies. Replies to SMTP commands are devised to ensure the synchronization of requests and actions in the process of mail transfer, and to guarantee that the sender-SMTP always knows the state of the receiver-SMTP. Every command must generate exactly one reply. The details of the command-reply sequence are made explicit in Section 5.3 on Sequencing and Section 5.4 State Diagrams. An SMTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is meant for the human user. It is intended that the three digits contain enough encoded information that the sender-SMTP need not examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be receiver-dependent and context dependent, so there are likely to be varying texts for each reply code. A discussion of the theory of reply codes is given in paragraph 8. Formally, a reply is defined to be the sequence: a three-digit code, <SP>, one line of text, and

MIL-STD-1781
10 May 1984

<CRLF>, or a multiline reply (as defined in Appendix E). Only the EXPN and HELP commands are expected to result in multiline replies in normal circumstances; however, multiline replies are allowed for any command.

6.2.1 Reply codes by function groups.

- a. 500 Syntax error, command unrecognized
[This may include errors such as command line too long]
- b. 501 Syntax error in parameters or arguments
- c. 502 Command not implemented
- d. 503 Bad sequence of commands
- e. 504 Command parameter not implemented
- f. 211 System status, or system help reply
- g. 214 Help message
[Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
- h. 220 <domain> Service ready
- i. 221 <domain> Service closing transmission channel
- j. 421 <domain> Service not available,
closing transmission channel
[This may be a reply to any command if the service knows it must shut down]
- k. 250 Requested mail action okay, completed
- l. 251 User not local; will forward to <forward-path>
- m. 450 Requested mail action not taken: mailbox unavailable
[e.g., mailbox busy]
- n. 550 Requested action not taken: mailbox unavailable
[e.g., mailbox not found, no access]
- o. 451 Requested action aborted: error in processing
- p. 551 User not local; please try <forward-path>
- q. 452 Requested action not taken: insufficient system storage
- r. 552 Requested mail action aborted: exceeded storage allocation

MIL-STD-1781

10 May 1984

- s. 553 Requested action not taken: mailbox name not allowed
[e.g., mailbox syntax incorrect]
- t. 354 Start mail input; end with <CRLF>.<CRLF>
- u. 554 Transaction failed

6.2.2 Numeric order list of reply codes.

- a. 211 System status, or system help reply
- b. 214 Help message
[Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
- c. 220 <domain> Service ready
- d. 221 <domain> Service closing transmission channel
- e. 250 Requested mail action okay, completed
- f. 251 User not local; will forward to <forward-path>
- g. 354 Start mail input; end with <CRLF>.<CRLF>
- h. 421 <domain> Service not available,
closing transmission channel
[This may be a reply to any command if the service knows it must shut down]
- i. 450 Requested mail action not taken: mailbox unavailable
[e.g., mailbox busy]
- j. 451 Requested action aborted: local error in processing
- k. 452 Requested action not taken: insufficient system storage
- l. 500 Syntax error, command unrecognized
[This may include errors such as command line too long]
- m. 501 Syntax error in parameters or arguments
- n. 502 Command not implemented
- o. 503 Bad sequence of commands
- p. 504 Command parameter not implemented
- q. 550 Requested action not taken: mailbox unavailable
[e.g., mailbox not found, no access]

MIL-STD-1781

10 May 1984

- r. 551 User not local; please try <forward-path>
- s. 552 Requested mail action aborted: exceeded storage allocation
- t. 553 Requested action not taken: mailbox name not allowed
[E.g., mailbox syntax incorrect]
- u. 554 Transaction failed

6.3 Sequencing of commands and replies. The communication between the sender and receiver is intended to be an alternating dialogue, controlled by the sender. As such, the sender issues a command and the receiver responds with a reply. The sender must wait for this response before sending further commands. One important reply is the connection greeting. Normally, a receiver will send a 220 "Service ready" reply when the connection is completed. The sender should wait for this greeting message before sending any commands. All the greeting type replies have the official name of the server host as the first word following the reply code. For example, 220 <SP> USC-ISIP.ARPA <SP> Service ready <CRLF>. Paragraph 6.3.1 lists alternative success and failure replies for each command. These must be strictly adhered to; a receiver may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

6.3.1 Command-reply sequences. Each command is listed with its possible replies. The prefixes used before the possible replies are "P" for preliminary (not used in SMTP), "I" for intermediate, "S" for success, "F" for failure, and "E" for error. The 421 reply (service not available, closing transmission channel) may be given to any command if the SMTP-receiver knows it must shut down. This listing forms the basis for the State Diagrams in paragraph 6.4.

CONNECTION ESTABLISHMENT

S: 220

F: 421

HELO

S: 250

E: 500, 501, 504, 421

MAIL

S: 250

F: 552, 451, 452

E: 500, 501, 421

RCPT

S: 250, 251

F: 550, 551, 552, 553, 450, 451, 452

E: 500, 501, 503, 421

DATA

I: 354 -> data -> S: 250

P: 552, 554, 451, 452

F: 451, 554

E: 500, 501, 503, 421

MIL-STD-1781

10 May 1984

RSET
S: 250
E: 500, 501, 504, 421

SEND
S: 250
F: 552, 451, 452
E: 500, 501, 502, 421

SGML
S: 250
F: 552, 451, 452
E: 500, 501, 502, 421

SAML
S: 250
F: 552, 451, 452
E: 500, 501, 502, 421

VRFY
S: 250, 251
F: 550, 551, 553
E: 500, 501, 502, 504, 421

EXPN
S: 250
F: 550
E: 500, 501, 502, 504, 421

HELP
S: 211, 214
E: 500, 501, 502, 504, 421

NOOP
S: 250
E: 500, 421

QUIT
S: 221
E: 500

TURN
S: 250
F: 502
E: 500, 503

6.4 State diagrams. The following are state diagrams for a simple SMTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of SMTP commands. The command groupings were determined by constructing a model for each command and then collecting together the commands with structurally identical models. For each command there are three possible outcomes: "success" (S), "failure" (F), and "error" (E). In the state diagrams below we use the symbol B for "begin," and the symbol W for "wait for reply."

6.4.1 SMTP commands. Figure 2 represents most of the SMTP commands.

MIL-STD-1781
10 May 1984

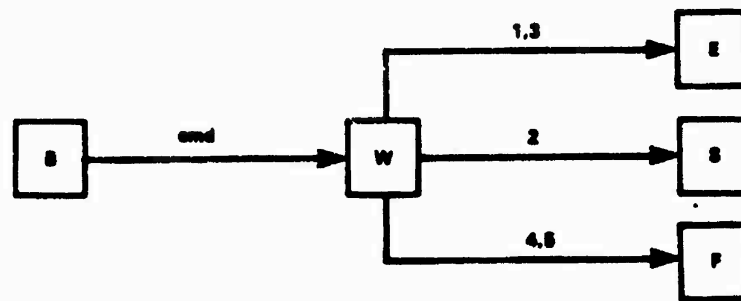


FIGURE 2. Representation of most of the SMTP commands.

Figure 2 models the commands:

HELO, MAIL, RCPT, RSET, SEND, SOML, SAML, VRFY, EXPN, HELP,
NOOP, QUIT, TURN.

6.4.2 The DATA command. Figure 3 models the DATA command.

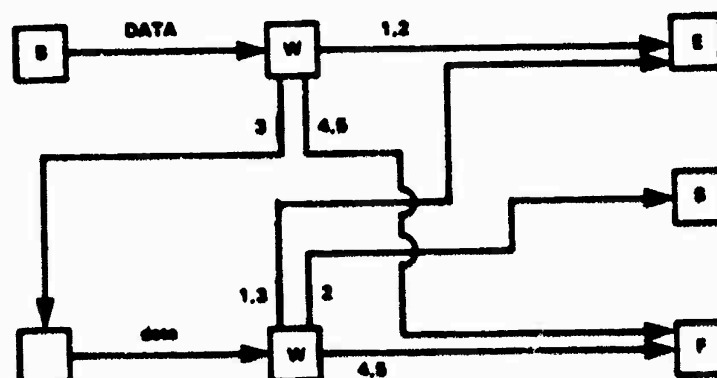


FIGURE 3. The DATA command.

Note that the "data" here is a series of lines sent from the sender to the receiver with no response expected until the last line is sent.

6.5 Details.

6.5.1 Minimum implementation. In order to make SMTP workable, the following minimum implementation is required for all receivers:

MIL-STD-1781

10 May 1984

COMMANDS -- HELO
MAIL
RCPT
DATA
RSET
NOOP
QUIT

6.5.2 Transparency. Without some provision for data transparency the character sequence "<CRLF>.<CRLF>" ends the mail text and cannot be sent by the user. In general, users are not aware of such "forbidden" sequences. To allow all user composed text to be transmitted transparently the following procedures are used.

- a. Before sending a line of mail text the sender-SMTP checks the first character of the line. If it is a period, one additional period is inserted at the beginning of the line.
- b. When a line of mail text is received by the receiver-SMTP it checks the line. If the line is composed of a single period it is the end of mail. If the first character is a period and there are other characters on the line, the first character is deleted.

6.5.2.1 ASCII characters. The mail data may contain any of the 128 ASCII characters. All characters are to be delivered to the recipient's mailbox including format effectors and other control characters. If the transmission channel provides an 8-bit byte (octets) data stream, the 7-bit ASCII codes are transmitted right justified in the octets with the high order bits cleared to zero. In some systems it may be necessary to transform the data as it is received and stored. This may be necessary for hosts that use a different character set than ASCII as their local character set, or that store data in records rather than strings. If such transforms are necessary, they must be reversible — especially if such transforms are applied to mail being relayed.

6.5.3 Sizes. There are several objects that have required minimum maximum sizes. That is, every implementation must be able to receive objects of at least these sizes, but must not send objects larger than these sizes. To the maximum extent possible, implementation techniques which impose no limits on the length of these objects should be used.

- a. User
The maximum total length of a user name is 64 characters.
- b. Domain
The maximum total length of a domain name or number is 64 characters.
- c. Path
The maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators).

MIL-STD-1781

10 May 1984

- d. **Command Line**
The maximum total length of a command line including the command word and the <CRLF> is 512 characters.
- e. **Reply Line**
The maximum total length of a reply line including the reply code and the <CRLF> is 512 characters.
- f. **Text Line**
The maximum total length of a text line including the <CRLF> is 1000 characters (but not counting the leading dot duplicated for transparency).
- g. **Recipients Buffer**
The maximum total number of recipients that must be buffered is 100 recipients.

6.5.3.1 Error reply codes. Errors due to exceeding these limits may be reported by using the reply codes, for example:

500 Line too long.

501 Path too long.

552 Too many recipients.

552 Too much mail data.

MIL-STD-1781
10 May 1984

7. SERVICES

7.1 Transmission Control Protocol (TCP) transport service. The Transmission Control Protocol (MIL-STD-1778) is mandatory for use in all DoD packet switched networks which connect or have the potential for utilizing connectivity across network or subnetwork boundaries.

7.1.1 Connection establishment. The SMTP transmission channel is a TCP connection established between the sender process port U and the receiver process port L. This single full duplex connection is used as the transmission channel. This protocol is assigned the service port 25 (31 octal), that is L = 25.

7.1.2 Data transfer. The TCP connection supports the transmission of 8-bit bytes. The SMTP data is 7-bit ASCII characters. Each character is transmitted as an 8-bit byte with the high-order bit cleared to zero.

7.2 X.25 transport service. It may be possible to use the X.25 service as directly provided by the Public Data Networks. However, it is suggested that a reliable end-to-end protocol such as TCP be used on top of X.25 connections.

MIL-STD-1781

10 May 1984

S: RSET
R: 250 OK

S: QUIT
R: 221 MIT-Multics.ARPA Service closing transmission channel

9.4 Relayed mail scenario.

9.4.1 Source host to relay host.

R: 220 USC-ISIE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-AI.ARPA
R: 250 USC-ISIE.ARPA

S: MAIL FROM:<JQP@MIT-AI.ARPA>
R: 250 OK

S: RCPT TO:<@USC-ISIE.ARPA:Jones@BBN-VAX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Date: 2 Nov 81 22:33:44
S: From: John Q. Public <JQP@MIT-AI.ARPA>
S: Subject: The Next Meeting of the Board
S: To: Jones@BBN-Vax.ARPA
S:
S: Bill:
S: The next meeting of the board of directors will be
S: on Tuesday.
S: John.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE.ARPA Service closing transmission channel

9.4.2 Relay host to destination host.

R: 220 BBN-VAX.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIE.ARPA
R: 250 BBN-VAX.ARPA

S: MAIL FROM:<@USC-ISIE.ARPA:JQP@MIT-AI.ARPA>
R: 250 OK

S: RCPT TO:<Jones@BBN-VAX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Received: from MIT-AI.ARPA by USC-ISIE.ARPA ;
2 Nov 81 22:40:10 UT

MIL-STD-1781
10 May 1984

9. SCENARIOS

9.1 Introduction. This section presents complete scenarios of several types of SMTP sessions.

9.2 A typical SMTP transaction scenario. This SMTP example shows mail sent by Smith at host USC-ISIF, to Jones, Green, and Brown at host BBN-UNIX. Here we assume that host USC-ISIF contacts host BBN-UNIX directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host BBN-UNIX.

R: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready

S: HELO USC-ISIF.ARPA

R: 250 BBN-UNIX.ARPA

S: MAIL FROM:<Smith@USC-ISIF.ARPA>

R: 250 OK

S: RCPT TO:<Jones@BBN-UNIX.ARPA>

R: 250 OK

S: RCPT TO:<Green@BBN-UNIX.ARPA>

R: 550 No such user here

S: RCPT TO:<Brown@BBN-UNIX.ARPA>

R: 250 OK

S: DATA

R: 354 Start mail input; end with <CRLF>.<CRLF>

S: ...etc. etc. etc.

S: ...etc. etc. etc.

S: .

R: 250 OK

S: QUIT

R: 221 BBN-UNIX.ARPA Service closing transmission channel

9.3 Aborted SMTP transaction scenario.

R: 220 MIT-Multics.ARPA Simple Mail Transfer Service Ready

S: HELO ISI-VAXA.ARPA

R: 250 MIT-Multics.ARPA

S: MAIL FROM:<Smith@ISI-VAXA.ARPA>

R: 250 OK

S: RCPT TO:<Jones@MIT-Multics.ARPA>

R: 250 OK

S: RCPT TO:<Green@MIT-Multics.ARPA>

R: 550 No such user here

MIL-STD-1781

10 May 1984

8.1.2 Values for the second digit. The second digit encodes responses in specific categories:

8.1.2.1 Syntax (x0z). These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, and unimplemented or superfluous commands.

8.1.2.2 Information (x1z). These are replies to requests for information, such as status or help.

8.1.2.3 Connections (x2z). These are replies referring to the transmission channel.

8.1.2.4 Unspecified (x3z).

8.1.2.5 Unspecified (x4z).

8.1.2.6 Mail system (x5z). These replies indicate the status of the receiver mail system vis-a-vis the requested transfer or other mail system action.

8.1.3 Values for the third digit. The third digit gives a finer gradation of meaning in each category specified by the second digit. The list of replies illustrates this. Each reply text is recommended rather than mandatory, and may even change according to the command with which it is associated. On the other hand, the reply codes must strictly follow the specifications in this section. Receiver implementations should not invent new codes for slightly different situations from the ones described here, but rather adapt codes already defined. For example, a command such as NOOP whose successful execution does not offer the sender-SMTP any new information will return a 250 reply. The response is 502 when the command requests an unimplemented non-site-specific action. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

8.1.3.1 Special format. The reply text may be longer than a single line; in these cases the complete text must be marked so the sender-SMTP knows when it can stop reading the reply. This requires a special format to indicate a multiple line reply. The format for multiline replies requires that every line, except the last, begin with the reply code, followed immediately by a hyphen, "--" (also known as minus), followed by text. The last line will begin with the reply code, followed immediately by <SP>, optionally some text, and <CRLF>. For example:

```
123 - First line
123 - Second line
123-234 Text beginning with numbers
123 - The last line
```

In many cases the sender-SMTP then simply needs to search for the reply code followed by <SP> at the beginning of a line, and ignore all preceding lines. In a few cases, there is important data for the sender in the reply "text." The sender will know these cases from the current context.

MIL-STD-1781

10 May 1984

8. THEORY OF REPLY CODES

8.1 Introduction. The three digits of the reply each have a special significance. The first digit denotes whether the response is good, bad or incomplete. An unsophisticated sender-SMTP will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A sender-SMTP that wants to know approximately what kind of error occurred (e.g., mail system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information.

8.1.1 Values for the first digit. There are five values for the first digit of the reply code.

8.1.1.1 Positive preliminary reply (1yz). The command has been accepted, but the requested action is being held in abeyance, pending confirmation of the information in this reply. The sender-SMTP should send another command specifying whether to continue or abort the action. SMTP does not have any commands that allow this type of reply, and so does not have the continue or abort commands.

8.1.1.2 Positive completion reply (2yz). The requested action has been successfully completed. A new request may be initiated.

8.1.1.3 Positive intermediate reply (3yz). The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The sender-SMTP should send another command specifying this information. This reply is used in command sequence groups.

8.1.1.4 Transient negative completion reply (4yz). The command was not accepted and the requested action did not occur. However, the error condition is temporary and the action may be requested again. The sender should return to the beginning of the command sequence (if any). It is difficult to assign a meaning to "transient" when two different sites (receiver- and sender-SMTPs) must agree on the interpretation. Each reply in this category might have a different time value, but the sender-SMTP is encouraged to try again. A rule of thumb to determine if a reply fits into the 4yz or the 5yz category (see below) is that replies are 4yz if they can be repeated without any change in command form or in properties of the sender or receiver. (E.g., the command is repeated identically and the receiver does not put up a new implementation.)

8.1.1.5 Permanent negative completion reply (5yz). The command was not accepted and the requested action did not occur. The sender-SMTP is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct the sender-SMTP to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered the account status).

MIL-STD-1781
10 May 1984

S: Date: 2 Nov 81 22:33:44
S: From: John Q. Public <JQP@MIT-AI.ARPA>
S: Subject: The Next Meeting of the Board
S: To: Jones@BBN-Vax.ARPA
S:
S: Bill:
S: The next meeting of the board of directors will be
S: on Tuesday.
S: John.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE.ARPA Service closing transmission channel

9.5 Verifying and sending scenario.

R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SEND FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...etc. etc. etc.
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel

9.6 Sending and mailing scenarios. First the user's name is verified, then an attempt is made to send to the user's terminal. When that fails, the message is mailed to the user's mailbox.

R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SEND FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

MIL-STD-1781 10 May 1984

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 450 User not active now

S: RSET
R: 250 OK

S: MAIL FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...etc. etc. etc.
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel

9.6.1 Doing the sending and mailing scenarios more efficiently.

R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SOHL FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 User not active now, so will do mail.

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...etc. etc. etc.
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel

9.7 Mailing list scenario. First each of two mailing lists are expanded in separate sessions with different hosts. Then the message is sent to everyone that appeared on either list (but no duplicates) via a relay host.

MIL-STD-1781

10 May 1984

9.7.1 Expanding the first list.

R: 220 MIT-AI.ARPA Simple Mail Transfer Service Ready
S: HELO SU-SCORE.ARPA
R: 250 MIT-AI.ARPA

S: EXPN Example-People
R: 250-<ABC@MIT-MC.ARPA>
R: 250-Fred Fonebone <Fonebone@USC-ISIQ.ARPA>
R: 250-Xenon Y. Zither <XYZ@MIT-AI.ARPA>
R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250-<joe@foo-unix.ARPA>
R: 250 <xyz@bar-unix.ARPA>

S: QUIT
R: 221 MIT-AI.ARPA Service closing transmission channel

9.7.2 Expanding the second list.

R: 220 MIT-MC.ARPA Simple Mail Transfer Service Ready
S: HELO SU-SCORE.ARPA
R: 250 MIT-MC.ARPA

S: EXPN Interested-Parties
R: 250-Al Calico <ABC@MIT-MC.ARPA>
R: 250-<XYZ@MIT-AI.ARPA>
R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250-<fred@BEN-UNIX.ARPA>
R: 250 <xyz@bar-unix.ARPA>

S: QUIT
R: 221 MIT-MC.ARPA Service closing transmission channel

9.7.3 Mailing to all via a relay host.

R: 220 USC-ISIE.ARPA Simple Mail Transfer Service Ready
S: HELO SU-SCORE.ARPA
R: 250 USC-ISIE.ARPA

S: MAIL FROM:<Account.Person@SU-SCORE.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:ABC@MIT-MC.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:Fonebone@USC-ISIQA.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:XYZ@MIT-AI.ARPA>
R: 250 OK
S: RCPT
TO:<@USC-ISIE.ARPA,@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:joe@FOO-UNIX.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:xyz@BAR-UNIX.ARPA>

MIL-STO-1781

10 May 1984

R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:fred@BBN-UNIX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...etc. etc. etc.
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE.ARPA Service closing transmission channel

9.8 Forwarding scenarios.

R: 220 USC-ISIF.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISIF.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<fred@USC-ISIF.ARPA>
R: 251 User not local; will forward to <Jones@USC-ISI.ARPA>

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...etc. etc. etc.
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIF.ARPA Service closing transmission channel

9.9 Reading and delivering the mail.

9.9.1 Trying the mailbox at the first host.

R: 220 USC-ISIF.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISIF.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<fred@USC-ISIF.ARPA>
R: 251 User not local; will forward to <Jones@USC-ISI.ARPA>

S: RSET
R: 250 OK

MIL-STD-1781
10 May 1984

S: QUIT
R: 221 USC-ISIF.ARPA Service closing transmission channel

9.9.2 Delivering the mail at the second host.

R: 220 USC-ISI.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISI.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<Jones@USC-ISI.ARPA>
R: OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...etc. etc. etc.
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISI.ARPA Service closing transmission channel

9.10 Too many recipients scenario.

R: 220 BERKELEY.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIF.ARPA
R: 250 BERKELEY.ARPA

S: MAIL FROM:<Postel@USC-ISIF.ARPA>
R: 250 OK

S: RCPT TO:<fabry@BERKELEY.ARPA>
R: 250 OK

S: RCPT TO:<eric@BERKELEY.ARPA>
R: 552 Recipient storage full, try again in another transaction

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...etc. etc. etc.
S: ...etc. etc. etc.
S: .
R: 250 OK

S: MAIL FROM:<Postel@USC-ISIF.ARPA>
R: 250 OK

S: RCPT TO:<eric@BERKELEY.ARPA>
R: 250 OK

MIL-STD-1781
10 May 1984

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...etc. etc. etc.
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 BERKELEY.ARPA Service closing transmission channel

Note that a real implementation must handle many recipients as specified in Section 4.5.3.

Custodians:

Army - CR
Navy - OM
Air Force - 90
Review Activities:
Army - SC, CR, AD
Navy - AS, YD, MC, OM, ND, NC, EC, SA
Air Force - 01, 02, 11, 13, 17, 99,

Preparing Activity:
DCA - DC

(Project IPSC-0195-02)
Other Interest:
NSA - NS
JTCC-TT

MIL-STD 1782

10 MAY 1984

MILITARY STANDARD

TELNET PROTOCOL



**NO DELIVERABLE DATA
REQUIRED BY THIS DOCUMENT**

IPSC/SJMC/TCTS

MIL-STD-1782
10 May 1984

DEPARTMENT OF DEFENSE
WASHINGTON, D.C. 20301

TELNET Protocol

MIL-STD-1782

1. This Military Standard is approved for use by all Departments and Agencies of the Department of Defense.
2. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to: Defense Communications Agency, ATTN: J110, 1860 Wiehle Avenue, Reston, Virginia 22090, by using the self-addressed Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document, or by letter.
3. Because of the rapid development in this standardization area, an alternative method of communication is offered. Forward responses using the MILNET to DCA-IAS @ DCA-EMS. Cooperation of the user is important to make this protocol meet Department of Defense needs.

MIL-STD-1782

10 May 1984

FOREWORD

This document specifies the TELNET protocol and a number of approved options. It provides a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation).

MIL-STD-1782
10 May 1984

CONTENTS

Paragraph		Page
1.	SCOPE- - - - -	1
1.1	Purpose- - - - -	1
1.2	Organization - - - - -	1
1.3	Application- - - - -	1
2.	REFERENCED DOCUMENTS - - - - -	2
2.1	Issues of documents- - - - -	2
2.2	Other publications - - - - -	2
3.	DEFINITIONS- - - - -	3
4.	GENERAL REQUIREMENTS - - - - -	4
4.1	Introduction - - - - -	4
4.2	General considerations - - - - -	4
4.2.1	Network virtual terminal (NVT) - - - - -	4
4.2.2	Principle of negotiated options- - - - -	4
4.2.3	Symmetry of the negotiation syntax - - - - -	5
4.2.4	Use of options - - - - -	5
4.2.5	Synopsis - - - - -	6
4.3	The network virtual terminal - - - - -	6
4.3.1	Transmission of data - - - - -	6
4.3.1.1	Accumulation of data - - - - -	6
4.3.1.2	TELNET Go Ahead (GA) command - - - - -	7
4.3.2	Transmission caution - - - - -	7
4.4	Standard representation of control functions - - - - -	8
4.4.1	Interrupt process (IP) - - - - -	8
4.4.2	Abort output (AO)- - - - -	8
4.4.3	Are you there (AYT)- - - - -	9
4.4.4	Erase character (EC) - - - - -	9
4.4.5	Erase line (EL)- - - - -	9
4.5	The TELNET Synch signal- - - - -	9
4.5.1	Sending several Synchs - - - - -	9
4.5.2	Interesting signals- - - - -	10
4.5.3	One effect of the Synch mechanism- - - - -	10
4.5.4	TELNET command requirement - - - - -	10
4.6	The NVT printer and keyboard - - - - -	10
4.6.1	NVT printer codes- - - - -	10
4.6.1.1	Example of code use- - - - -	11
4.6.1.2	ASCII code generation- - - - -	12
4.6.1.3	Additional codes - - - - -	12
4.6.1.3.1	Synch- - - - -	12
4.6.1.3.2	Break (BRK)- - - - -	12
4.6.1.3.3	Interrupt process (IP) - - - - -	12
4.6.1.3.4	Abort output (AO)- - - - -	12
4.6.1.3.5	Are you there (AYT)- - - - -	12
4.6.1.3.6	Erase character (EC) - - - - -	12
4.6.1.3.7	Erase line (EL)- - - - -	13
4.6.1.3.8	Intent of additional codes - - - - -	13

MIL-STD-1782
10 May 1984

CONTENTS - Continued

	<u>Page</u>
Paragraph 4.7	TELNET command structure - - - - - 13
4.7.1	TELNET commands defined- - - - - 13
4.8	Connection establishment - - - - - 14
4.8.1	Port assignment- - - - - 14

APPENDICES

Appendix A.	TELNET BINARY TRANSMISSION - - - - - 15
10.	Command name and code- - - - - 15
20.	Command meanings - - - - - 15
20.1	IAC WILL TRANSMIT - BINARY - - - - - 15
20.2	IAC WON'T TRANSMIT - BINARY- - - - - 15
20.3	IAC DO TRANSMIT - BINARY - - - - - 15
20.4	IAC DON'T TRANSMIT - BINARY- - - - - 15
30.	Default- - - - - 15
40.	Motivation for the option- - - - - 16
50.	Description of the option- - - - - 16
60.	Implementation suggestions - - - - - 16
70.	Binary transmission mode - - - - - 17
70.1	Binary transmission from a terminal- - - - - 17
70.2	Binary transmission to a process - - - - - 17
70.3	Binary transmission from a process - - - - - 17
70.4	Binary transmission to a terminal- - - - - 17
Appendix B.	TELNET ECHO OPTION - - - - - 18
10.	Command name and code- - - - - 18
20.	Command meanings - - - - - 18
20.1	IAC WILL ECHO- - - - - 18
20.2	IAC WON'T ECHO - - - - - 18
20.3	IAC DO ECHO- - - - - 18
20.4	IAC DON'T ECHO - - - - - 18
30.	Default- - - - - 18
40.	Motivation for the option- - - - - 18
50.	Description of the option- - - - - 19
60.	Implementation of the option - - - - - 20
60.1	A sample implementation of the option- - - - - 20
Appendix C.	TELNET SUPPRESS GO AHEAD OPTION- - - - - 22
10.	Command name and code- - - - - 22
20.	Command meanings - - - - - 22
20.1	IAC WILL SUPPRESS-TO-AHEAD - - - - - 22
20.2	IAC WON'T SUPPRESS-GO-AHEAD- - - - - 22
20.3	IAC DO SUPPRESS-GO-AHEAD - - - - - 22
20.4	IAC DON'T SUPPRESS-GO-AHEAD- - - - - 22
30.	Default- - - - - 22
40.	Motivation for the option- - - - - 22
50.	Description of the option- - - - - 22
60.	Implementation considerations- - - - - 23

MIL-STD-1782
10 May 1984

CONTENTS - Continued

		<u>Page</u>
Appendix D.	TELNET STATUS OPTION - - - - -	24
10.	Command name and code- - - - -	24
20.	Command meanings - - - - -	24
20.1	IAC WILL STATUS - - - - -	24
20.2	IAC WON'T STATUS - - - - -	24
20.3	IAC DO STATUS- - - - -	24
20.4	IAC DON'T STATUS - - - - -	24
20.5	IAC SB STATUS SEND IAC SE- - - - -	24
20.6	IAC SB STATUS IS...IAC SE- - - - -	24
30.	Default- - - - -	24
40.	Motivation for the option- - - - -	24
50.	Description of the option- - - - -	24
Appendix E.	TELNET TIMING MARK OPTION- - - - -	25
10.	Command name and code- - - - -	25
20.	Command meanings - - - - -	25
20.1	IAC WILL TIMING-MARK - - - - -	25
20.2	IAC WON'T TIMING-MARK- - - - -	25
20.3	IAC DO TIMING-MARK - - - - -	25
20.4	IAC DON'T TIMING-MARK - - - - -	25
30.	Default - - - - -	25
40.	Motivation for the option - - - - -	25
50.	Examples of option application- - - - -	25
60.	Description of the option - - - - -	25
70.	Three typical applications- - - - -	27
70.1	Round-trip delay- - - - -	27
70.2	Resynchronization - - - - -	27
70.3	Dual resynchronization- - - - -	27
Appendix F.	TELNET EXTENDED OPTIONS - LIST OPTION - - - - -	28
10.	Command name and code - - - - -	28
20.	Command meanings- - - - -	28
20.1	IAC WILL EXOPL- - - - -	28
20.2	IAC WON'T EXOPL - - - - -	28
20.3	IAC DO EXOPL- - - - -	28
20.4	IAC DON'T EXOPL - - - - -	28
20.5	IAC SB EXOPL subcommand - - - - -	28
30.	Default - - - - -	28
40.	Motivation for the option - - - - -	28
50.	Description of the option - - - - -	28

MIL-STD-1782

10 May 1984

FIGURES

			<u>Page</u>
Figure	1	Five reasonable modes of operation for	
		echoing on a connection pair - - - - -	19
	2	Synchronization of processes- - - - -	26

MIL-STD-1782
10 May 1984

1. SCOPE

1.1 Purpose. This standard establishes criteria for the TELNET Protocol which supports the standard method of interfacing terminal devices and terminal-oriented processes to each other.

1.2 Organization. This standard introduces the TELNET Protocols role, defines the services provided to users, and specifies the mechanisms needed to support those services. This standard also includes an appendix of options which can be implemented in the TELNET Protocol and a glossary of terms and abbreviations.

1.3 Application. This TELNET Protocol is approved for use in all DoD packet switching networks which connect or have the potential for utilizing connectivity across network and subnetwork boundaries and which require a virtual terminal service. The term network as used herein includes Local Area Networks.

MIL-STD-1782
10 May 1984

2. REFERENCED DOCUMENTS

2.1 Issues of documents. The following documents of the issue in effect on date of invitation for bids or request for proposal, form a part of this standard to the extent specified herein.

STANDARDS

FEDERAL

FED-STD-1037

Glossary of Telecommunications Terms

MILITARY

MIL-STD-1778

Transmission Control Protocol

2.2 Other publications. The following documents form a part of this standard to the extent specified herein. Unless otherwise indicated, the issue in effect on date of invitation for bids or request for proposal shall apply. (The provisions of this paragraph are under consideration.)

MIL-STD-1782

10 May 1984

3. DEFINITIONS

3.1 Definition of terms. The definition of terms used in this standard shall comply with FED-STD-1037. Terms and definitions unique to MIL-STD-1782 are contained herein. (The provisions of this paragraph are under consideration.)

3.2 Definitions of acronyms used in this standard. The following acronyms used in this Military Standard are defined as follows:

- a. AO - Abort Output Command
- b. AYT - Are You There Command
- c. BS - Back Space
- d. CR - Carriage Return
- e. DDN - Defense Data Network
- f. DM - Data Mark
- g. DoD - Department of Defense
- h. DODIIS - DoD Intelligence Information System
- i. EC - Erase Character Command
- j. EL - Erase Line Command
- k. FF - Form Feed
- l. GA - Go-Ahead Command
- m. HT - Horizontal Tab
- n. IAC - Interpret As Command
- o. IBM - International Business Machines, Inc.
- p. IP - Interrupt Process Command
- q. LF - Line Feed
- r. NVT - Network Virtual Terminal
- s. TCP - Transmission Control Protocol
- t. TELNET - Telecommunications Network
- u. VT - Vertical Tab
- v. ASCII - American Standard Code for Information Interchange

MIL-STD-1782

10 May 1984

4. GENERAL REQUIREMENTS

4.1 Introduction. The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation). The Appendices to this volume contain DoD approved and supported TELNET options. It is recommended, but not mandatory, that these options be implemented as useful adjuncts to the TELNET protocol. If implemented, they are required to be implemented as published herein.

4.2 General considerations. A TELNET connection is a Transmission Control Protocol (TCP) connection used to transmit data with interspersed TELNET control information. The TELNET Protocol is built upon three main ideas: first, the concept of a Network Virtual Terminal; second, the principle of negotiated options; and third, a symmetric view of terminals and processes.

4.2.1 Network Virtual Terminal (NVT). When a TELNET connection is first established, each end is assumed to originate and terminate at a Network Virtual Terminal (NVT). An NVT is an imaginary device which provides a standard, network-wide, intermediate representation of a canonical terminal. This eliminates the need for "server" and "user" hosts to keep information about the characteristics of each other's terminals and terminal handling conventions. All hosts, both user and server, map their local device characteristics and conventions so as to appear to be dealing with an NVT over the network, and each can assume a similar mapping by the other party. The NVT is intended to strike a balance between being overly restricted (not providing hosts a rich enough vocabulary for mapping into their local character sets), and being overly inclusive (penalizing users with modest terminals). The "user" host is the host to which the physical terminal is normally attached, and the "server" host is the host which is normally providing some service. As an alternate point of view, applicable even in terminal-to-terminal or process-to-process communications, the "user" host is the host which initiated the communication.

4.2.2 Principle of negotiated options. The principle of negotiated options takes cognizance of the fact that many hosts will wish to provide additional services over and above those available within an NVT, and many users will have sophisticated terminals and would like to have elegant, rather than minimal, services. Independent of, but structured within the TELNET Protocol are various "options" that will be sanctioned and may be used with the "DO, DON'T, WILL, WON'T" structure (discussed below) to allow user and server to agree to use a more elaborate (or perhaps just different) set of conventions for their TELNET connection. Such options could include changing the character set, the echo mode, etc. The basic strategy for setting up the use of options is to have either party (or both) initiate a request that some option take effect. The other party may then either accept or reject the request. If the request is accepted the option immediately takes effect; if it is rejected the associated aspect of the connection remains as specified for an NVT. Clearly, a party may always refuse a request to enable, and must never refuse

MIL-STD-1782
10 May 1984

a request to disable some option since all parties must be prepared to support the NVT. The syntax of option negotiation has been set up so that if both parties request an option simultaneously, each will see the other's request as the positive acknowledgment of its own.

4.2.3 Symmetry of the negotiation syntax. The symmetry of the negotiation syntax can potentially lead to nonterminating acknowledgment loops -- each party seeing the incoming commands not as acknowledgments but as new requests which must be acknowledged. To prevent such loops, the following rules prevail:

- a. Parties may only request a change in option status; i.e., a party may not send out a "request" merely to announce what mode it is in.
- b. If a party receives what appears to be a request to enter some mode it is already in, the request should not be acknowledged. This non-response is essential to prevent endless loops in the negotiation. It is required that a response be sent to requests for a change of mode -- even if the mode is not changed.
- c. Whenever one party sends an option command to a second party, whether as a request or an acknowledgment, and use of the option will have any effect on the processing of the data being sent from the first party to the second, then the command must be inserted in the data stream at the point where it is desired that it take effect. Some time will elapse between the transmission of a request and the receipt of an acknowledgment, which may be negative. Thus, a host may wish to buffer data, after requesting an option, until it learns whether the request is accepted or rejected, in order to hide the "uncertainty period" from the user.

4.2.4 Use of options. Option requests are likely to flurry back and forth when a TELNET connection is first established, as each party attempts to get the best possible service from the other party. Beyond that, however, options can be used to dynamically modify the characteristics of the connection to suit changing local conditions. For example, the NVT, as will be explained later, uses a transmission discipline well suited to the many "line at a time" applications such as BASIC, but poorly suited to the many "character at a time" applications such as NLS. A server might elect to devote the extra processor overhead required for a "character at a time" discipline when it was suitable for the local process and would negotiate an appropriate option. However, rather than then being permanently burdened with the extra processing overhead, it could switch (i.e., negotiate) back to NVT when the detailed control was no longer necessary. It is possible for requests initiated by processes to stimulate a nonterminating request loop if the process responds to a rejection by merely re-requesting the option. To prevent such loops from occurring, rejected requests should not be repeated until something changes. Operationally, this can mean the process is running a different program, or the user has given another command, or whatever makes sense in the context of the given process and the given option. A

MIL-STD-1782

10 May 1984

good rule of thumb is that a re-request should only occur as a result of subsequent information from the other end of the connection or when demanded by local human intervention. Option designers should not feel constrained by the somewhat limited syntax available for option negotiation. The intent of the simple syntax is to make it easy to have options -- since it is correspondingly easy to profess ignorance about them. If some particular option requires a richer negotiation structure than possible within "DO, DON'T, WILL, WON'T", the proper tack is to use "DO, DON'T, WILL, WON'T" to establish that both parties understand the option, and once this is accomplished a more exotic syntax can be used freely. For example, a party might send a request to alter (establish) line length. If it is accepted, then a different syntax can be used for actually negotiating the line length -- such a "sub-negotiation" might include fields for minimum allowable, maximum allowable and desired line lengths. The important concept is that such expanded negotiations should never begin until some prior (standard) negotiation has established that both parties are capable of parsing the expanded syntax.

4.2.5 Synopsis. In summary, WILL XXX is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and DON'T XXX being its positive and negative acknowledgments; similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing option XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the NVT is what is left when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, all hosts may implement their TELNET processes to be totally unaware of options that are not supported, simply returning a rejection to (i.e., refusing) any option request that cannot be understood. As much as possible, the TELNET protocol has been made server-user symmetrical so that it easily and naturally covers the user-user (linking) and server-server (cooperating processes) cases. It is hoped, but not absolutely required, that options will further this intent. In any case, it is explicitly acknowledged that symmetry is an operating principle rather than an ironclad rule. Standard TELNET options referenced in this document are contained in Appendices A-F.

4.3 The Network Virtual Terminal. The Network Virtual Terminal (NVT) is a bi-directional character device. The NVT has a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data which is sent over the TELNET connection and, if "echoes" are desired, to the NVT's printer as well. "Echoes" will not be expected to traverse the network (although options exist to enable a "remote" echoing mode of operation, no host is required to implement this option). The code set is seven-bit USASCII in an eight-bit field, except as modified herein. Any code conversion and timing considerations are local problems and do not affect the NVT.

4.3.1 Transmission of data. Although a TELNET connection through the network is intrinsically full duplex, the NVT is to be viewed as a half-duplex device operating in a line-buffered mode. That is, unless and until options are negotiated to the contrary, the following default conditions pertain to the transmission of data over the TELNET connection:

MIL-STD-1782
10 May 1984

4.3.1.1 Accumulation of data. Insofar as the availability of local buffer space permits, data should be accumulated in the host where it is generated until a complete line of data is ready for transmission, or until some locally-defined explicit signal to transmit occurs. This signal could be generated either by a process or by a human user. The motivation for this rule is the high cost, to some hosts, of processing network input interrupts, coupled with the default NVT specification that "echoes" do not traverse the network. Thus, it is reasonable to buffer some amount of data at its source. Many systems take some processing action at the end of each input line (even line printers or card punches frequently tend to work this way), so the transmission should be triggered at the end of a line. On the other hand, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line; therefore implementers are cautioned to provide methods of locally signaling that all buffered data should be transmitted immediately.

4.3.1.2 TELNET Go Ahead (GA) command. When a process has completed sending data to an NVT printer and has no queued input from the NVT keyboard for further processing (i.e., when a process at one end of a TELNET connection cannot proceed without input from the other end), the process must transmit the TELNET Go Ahead (GA) command. This rule is not intended to require that the TELNET GA command be sent from a terminal at the end of each line, since server hosts do not normally require a special signal (in addition to end-of-line or other locally-defined characters) in order to commence processing. Rather, the TELNET GA is designed to help a user's local host operate a physically half duplex terminal which has a "lockable" keyboard such as the IBM 2741. A description of this type of terminal may help to explain the proper use of the GA command. The terminal-computer connection is always under control of either the user or the computer. Neither can unilaterally seize control from the other; rather the controlling end must relinquish its control explicitly. At the terminal end, the hardware is constructed so as to relinquish control each time that a "line" is terminated (i.e., when the "New Line" key is typed by the user). When this occurs, the attached (local) computer processes the input data, decides if output should be generated, and if not returns control to the terminal. If output should be generated, control is retained by the computer until all output has been transmitted. The difficulties of using this type of terminal through the network should be obvious. The "local" computer is no longer able to decide whether to retain control after seeing an end-of-line signal or not; this decision can only be made by the "remote" computer which is processing the data. Therefore, the TELNET GA command provides a mechanism whereby the "remote" (server) computer can signal the "local" (user) computer that it is time to pass control to the user of the terminal. It should be transmitted at those times, and only at those times, when the user should be given control of the terminal. Note that premature transmission of the GA command may result in the blocking of output, since the user is likely to assume that the transmitting system has paused, and therefore he will fail to turn the line around manually.

MIL-STD-1782

10 May 1984

4.3.2 Transmission caution. The foregoing, of course, does not apply to the user-to-server direction of communication. In this direction, GAs may be sent at any time, but need not ever be sent. Also, if the TELNET connection is being used for process-to-process communication, GAs need not be sent in either direction. Finally, for terminal-to-terminal communication, GAs may be required in neither, one, or both directions. If a host plans to support terminal-to-terminal communication it is suggested that the host provide the user with a means of manually signaling that it is time for a GA to be sent over the TELNET connection; this, however, is not a requirement on the implementer of a TELNET process. The symmetry of the TELNET model requires an NVT at each end of the TELNET connection, at least conceptually.

4.4 Standard representation of control functions. As stated in paragraph 4.1, the primary goal of the TELNET protocol is the provision of a standard interfacing of terminal devices and terminal-oriented processes through the network. Early experiences with this type of interconnection have shown that certain functions are implemented by most servers, but that the methods of invoking these functions differ widely. For a human user who interacts with several server systems, these differences are highly frustrating. TELNET, therefore, defines a standard representation for five of these functions, as described below. These standard representations have standard, but not required, meanings (with the exception that the Interrupt Process (IP) function may be required by other protocols which use TELNET); that is, a system which does not provide the function to local users need not provide it to network users and may treat the standard representation for the function as a No-operation. On the other hand, a system which does provide the function to a local user is obliged to provide the same function to a network user who transmits the standard representation for the function.

4.4.1 Interrupt process (IP). Many systems provide a function which suspends, interrupts, aborts, or terminates the operation of a user process. This function is frequently used when a user believes his process is in an unending loop, or when an unwanted process has been inadvertently activated. IP is the standard representation for invoking this function. It should be noted by implementers that IP may be required by other protocols which use TELNET, and therefore should be implemented if these other protocols are to be supported.

4.4.2 Abort output (AO). Many systems provide a function which allows a process, which is generating output, to run to completion (or to reach the same stopping point it would reach if running to completion) but without sending the output to the user's terminal. Further, this function typically clears any output already produced but not yet actually printed (or displayed) on the user's terminal. AO is the standard representation for invoking this function. For example, some subsystem might normally accept a user's command, send a long text string to the user's terminal in response, and finally signal readiness to accept the next command by sending a "prompt" character (preceded by <CR><LF>) to the user's terminal. If the AO were received during the transmission of the text string, a reasonable implementation would be to suppress the remainder of the text string, but transmit the prompt character and the preceding <CR><LF>. (This is possibly in distinction to the action which might be taken if an IP were received; the IP might

MIL-STD-1782
10 May 1984

cause suppression of the text string and an exit from the subsystem.) It should be noted, by server systems which provide this function, that there may be buffers external to the system (in the network and the user's local host) which should be cleared; the appropriate way to do this is to transmit the "Synch" signal (described below) to the user system.

4.4.3 Are you there (AYT). Many systems provide a function which provides the user with some visible (e.g., printable) evidence that the system is still up and running. This function may be invoked by the user when the system is unexpectedly "silent" for a long time, because of the unanticipated (by the user) length of a computation, an unusually heavy system load, etc. AYT is the standard representation for invoking this function.

4.4.4 Erase character (EC). Many systems provide a function which deletes the last preceding undeleted character or "print position" from the stream of data being supplied by the user. A "print position" may contain several characters which are a result of overstrikes, or of sequences such as <char1> BS <char2>... This function is typically used to edit keyboard input when typing mistakes are made. EC is the standard representation for invoking this function.

4.4.5 Erase line (EL). Many systems provide a function which deletes all the data in the current "line" of input. This function is typically used to edit keyboard input. EL is the standard representation for invoking this function.

4.5 The TELNET "Synch" signal. Most time-sharing systems provide mechanisms which allow a terminal user to regain control of a "runaway" process; the IP and AO functions described above are examples of these mechanisms. Such systems, when used locally, have access to all of the signals supplied by the user, whether these are normal characters or special "out of band" signals such as those supplied by the teletype "BREAK" key or the IBM 2741 "ATTN" key. This is not necessarily true when terminals are connected to the system through the network; the network's flow control mechanisms may cause such a signal to be buffered elsewhere, for example in the user's host. To counter this problem, the TELNET "Synch" mechanism is introduced. A Synch signal consists of a TCP Urgent notification, coupled with the TELNET command DATA MARK. The Urgent notification, which is not subject to the flow control pertaining to the TELNET connection, is used to invoke special handling of the data stream by the process which receives it. In this mode, the data stream is immediately scanned for "interesting" signals as defined below, discarding intervening data. The TELNET command DATA MARK (DM) is the synchronizing mark in the data stream which indicates that any special signal has already occurred and the recipient can return to normal processing of the data stream. The Synch is sent via the TCP send operation with the Urgent flag set and the DM as the last (or only) data octet.

MIL-STD-1782
10 May 1984

4.5.1 Sending several Synchs. When several Synchs are sent in rapid succession, the Urgent notifications may be merged. It is not possible to count Urgents since the number received will be less than or equal the number sent. When in normal mode, a DM is a no operation; when in urgent mode, it signals the end of the urgent processing. If TCP indicates the end of Urgent data before the DM is found, TELNET should continue the special handling of the data stream until the DM is found. If TCP indicates more Urgent data after the DM is found, it can only be because of a subsequent Synch. TELNET should continue the special handling of the data stream until another DM is found.

4.5.2 Interesting signals. "Interesting" signals are defined to be: the TELNET standard representations of IP, AO, and AYT (but not EC or EL); the local analogs of these standard representations (if any); all other TELNET commands; other site-defined signals which can be acted on without delaying the scan of the data stream.

4.5.3 One effect of the Synch mechanism. Since one effect of the Synch mechanism is the discarding of essentially all characters (except TELNET commands) between the sender of the Synch and its recipient, this mechanism is specified as the standard way to clear the data path when that is desired. For example, if a user at a terminal causes an AO to be transmitted, the server which receives the AO (if it provides that function at all) should return a Synch to the user.

4.5.4 TELNET command requirement. Just as the TCP Urgent notification is needed at the TELNET level as an out-of-band signal, so other protocols which make use of TELNET may require a TELNET command which can be viewed as an out-of-band signal at a different level. By convention the sequence [IP, Synch] is to be used as such a signal. For example, suppose that some other protocol, which uses TELNET, defines the character string STOP analogously to the TELNET command AO. Imagine that a user of this protocol wishes a server to process the STOP string, but the connection is blocked because the server is processing other commands. The user should instruct his system to:

- a. Send the TELNET IP character;
- b. Send the TELNET Synch sequence, that is: Send the Data Mark (DM) as the only character in a TCP urgent mode send operation;
- c. Send the character string STOP; and
- d. Send the other protocol's analog of the TELNET DM, if any.

The user (or process acting on his behalf) must transmit the TELNET SYNCH sequence of step b above to ensure that the TELNET IP gets through to the server's TELNET interpreter. The Urgent should wake up the TELNET process; the IP should wake up the next higher level process.

4.6 The NVT printer and keyboard.

MIL-STD-1782

10 May 1984

4.6.1 NVT printer codes. The NVT printer has an unspecified carriage width and page length and can produce representations of all 95 ASCII graphics (codes 32 through 126). Of the 33 ASCII control codes (0 through 31 and 127), and the 128 uncovered codes (128 through 255), the following have specified meaning to the NVT printer:

	NAME	CODE	MEANING
a.	NULL (NUL)	0	No Operation
b.	Line Feed (LF)	10	Moves the printer to the next print line, keeping the same horizontal position.
c.	Carriage Return (CR)	13	Moves the printer to the left margin of the current line.

In addition, the following codes shall have defined, but not required, effects on the NVT printer. Neither end of a TELNET connection may assume that the other party will take, or will have taken, any particular action upon receipt or transmission of the following:

	NAME	CODE	MEANING
d.	BELL (BEL)	7	Produces an audible or visible signal (which does NOT move the print head).
e.	Back Space (BS)	8	Moves the print head one character position towards the left margin.
f.	Horizontal Tab (HT)	9	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
g.	Vertical Tab (VT)	11	Moves the printer to the next vertical tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
h.	Form Feed (FF)	12	Moves the printer to the top of the next page, keeping the same horizontal position.

All remaining codes do not cause the NVT printer to take any action.

MIL-STD-1782
10 May 1984

4.6.1.1 Example of code use. The sequence "CR LF", as defined, will cause the NVT to be positioned at the left margin of the next print line (as would, for example, the sequence "LF CR"). However, many systems and terminals do not treat CR and LF independently, and will have to go to some effort to simulate their effect. (For example, some terminals do not have a CR independent of the LF, but on such terminals it may be possible to simulate a CR by backspacing.) Therefore, the sequence "CR LF" must be treated as a single "new line" character and used whenever their combined action is intended; the sequence "CR NUL" must be used where a carriage return alone is actually desired; and the CR character must be avoided in other contexts. This rule gives assurance to systems which must decide whether to perform a "new line" function or a multiple-backspace that the TELNET stream contains a character following a CR that will allow a rational decision. Note that "CR LF" or "CR NUL" is required in both directions (in the default ASCII mode), to preserve the symmetry of the NVT model. Even though it may be known in some situations (e.g., with remote echo and suppress go ahead options in effect) that characters are not being sent to an actual printer, nonetheless, for the sake of consistency, the protocol requires that a NUL be inserted following a CR not followed by a LF in the data stream. The converse of this is that a NUL received in the data stream after a CR (in the absence of options negotiations which explicitly specify otherwise) should be stripped out prior to applying the NVT to local character set mapping.

4.6.1.2 ASCII code generation. The NVT keyboard has keys, or key combinations, or key sequences, for generating all 128 ASCII codes. Note that although many have no effect on the NVT printer, the NVT keyboard is capable of generating them.

4.6.1.3 Additional codes. In addition to these codes, the NVT keyboard shall be capable of generating the following additional codes which, except as noted, have defined, but not required, meanings. The actual code assignments for these "characters" are in the TELNET Command section, because they are viewed as being, in some sense, generic and should be available even when the data stream is interpreted as being some other character set.

4.6.1.3.1 Synch. This key allows the user to clear his data path to the other party. The activation of this key causes a DM (see command section) to be sent in the data stream and a TCP Urgent notification is associated with it. The pair DM-Urgent is to have required meaning as defined previously.

4.6.1.3.2 Break (BRK). This code is provided because it is a signal outside the ASCII set which is currently given local meaning within many systems. It is intended to indicate that the Break Key or the Attention Key was hit. Note, however, that this is intended to provide a 129th code for systems which require it, not as a synonym for the IP standard representation.

4.6.1.3.3 Interrupt process (IP). Suspend, interrupt, abort or terminate the process to which the NVT is connected. Also, part of the out-of-band signal for other protocols which use TELNET.

MIL-STD-1782

10 May 1984

4.6.1.3.4 Abort output (AO). Allow the current process to (appear to) run to completion, but do not send its output to the user. Also, send a Synch to the user.

4.6.1.3.5 Are you there (AYT). Send back to the NVT some visible (i.e., printable) evidence that the AYT was received.

4.6.1.3.6 Erase character (EC). The recipient should delete the last preceding undeleted character or "print position" from the data stream.

4.6.1.3.7 Erase line (EL). The recipient should delete characters from the data stream back to, but not including, the last "CR LF" sequence sent over the TELNET connection.

4.6.1.3.8 Intent of additional codes. The spirit of these "extra" keys, and also the printer format effectors, is that they should represent a natural extension of the mapping that already must be done from "NVT" into "local". Just as the NVT data byte 68 (104 octal) should be mapped into whatever the local code for "uppercase D" is, so the EC character should be mapped into whatever the local "Erase Character" function is. Further, just as the mapping for 124 (174 octal) is somewhat arbitrary in an environment that has no "vertical bar" character, the EL character may have a somewhat arbitrary mapping (or none at all) if there is no local "Erase Line" facility. Similarly for format effectors: if the terminal actually does have a "Vertical Tab", then the mapping for VT is obvious, and only when the terminal does not have a vertical tab should the effect of VT be unpredictable.

4.7 TELNET command structure. All TELNET commands consist of at least a two byte sequence: the "Interpret as Command" (IAC) escape character followed by the code for the command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option referenced. This format was chosen so that as more comprehensive use of the "data space" is made -- by negotiations from the basic NVT, of course -- collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and inefficiency, of "escaping" the data bytes into the stream. With the current set-up, only the IAC need be doubled to be sent as data, and the other 255 codes may be passed transparently.

4.7.1 TELNET commands defined. The following are the defined TELNET commands. Note that these codes and code sequences have the indicated meaning only when immediately preceded by an IAC.

CODE		
a. SE	240	End of subnegotiation parameters.
b. NOP	241	No operation.
c. Data Mark	242	The data stream portion of a Synch. This should always be accompanied by a TCP Urgent notification.

MIL-STD-1782

10 May 1984

d. Break	243	NVT character BRK.
e. Interrupt Process	244	The function IP.
f. Abort output	245	The function AO.
g. Are You There	246	The function AYT.
h. Erase character	247	The function EC.
i. Erase Line	248	The function EL.
j. Go ahead	249	The GA signal.
k. SB	250	Indicates that what follows is subnegotiation of the indicated option.
l. WILL (option code)	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option.
m. WON'T (option code)	252	Indicates the refusal to perform, or continue performing, the indicated option.
n. DO (option code)	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option.
o. DON'T (option code)	254	Indicates the demand that the other party stop performing, or confirmation that you are no longer expecting the other party to perform, the indicated option.
p. IAC	255	Data Byte 255.

4.8 Connection establishment. The TELNET TCP connection is established between the user's port U and the server's port L. The server listens on its well known port L for such connections. Since a TCP connection is full duplex and identified by the pair of ports, the server can engage in many simultaneous connections involving its port L and different user ports U.

4.8.1 Port assignment. When used for remote user access to service hosts (i.e., remote terminal access) this protocol is assigned server port 23 (27 octal). That is L=23.

MIL-STD-1782
10 May 1984

Custodians:

Army - CR
Navy - OM
Air Force - 90

Preparing Activity:

DCA - DC

(Project IPSC-0176-01)

Review Activities:

Army - SC, CR, AD
Navy - AS, YD, MC, OM, ND, NC, EC, SA
Air Force - 01, 02, 11, 13, 17, 99, 90

Other Interest:

NSA - NS
JTCCO-TT

MIL-STD-1782

10 May 1984

APPENDIX A

TELNET BINARY TRANSMISSION

10. Command name and code.

TRANSMIT-BINARY 0

20. Command meanings.

20.1 IAC WILL TRANSMIT-BINARY. The sender of this command REQUESTS permission to begin transmitting, or confirms that it will now begin transmitting characters which are to be interpreted as 8 bits of binary data by the receiver of the data.

20.2 IAC WON'T TRANSMIT-BINARY. If the connection is already being operated in binary transmission mode, the sender of this command DEMANDS to begin transmitting data characters which are to be interpreted as standard NVT ASCII characters by the receiver of the data. If the connection is not already being operated in binary transmission mode, the sender of this command REFUSES to begin transmitting characters which are to be interpreted as binary characters by the receiver of the data (i.e., the sender of the data demands to continue transmitting characters in its present mode). A connection is being operated in binary transmission mode only when one party has requested it and the other has acknowledged it.

20.3 IAC DO TRANSMIT-BINARY. The sender of this command REQUESTS that the sender of the data start transmitting, or confirms that the sender of data is expected to transmit, characters which are to be interpreted as 8 bits of binary data (i.e., by the party sending this command).

20.4 IAC DON'T TRANSMIT-BINARY. If the connection is already being operated in binary transmission mode, the sender of this command DEMANDS that the sender of the data start transmitting characters which are to be interpreted as standard NVT ASCII characters by the receiver of the data (i.e., the party sending this command). If the connection is not already being operated in binary transmission mode, the sender of this command DEMANDS that the sender of data continue transmitting characters which are to be interpreted in the present mode. A connection is being operated in binary transmission mode only when one party has requested it and the other has acknowledged it.

30. Default.

WON'T TRANSMIT-BINARY

DON'T TRANSMIT-BINARY

The connection is not operated in binary mode.

MIL-STD-1782
10 May 1984

40. Motivation for the option. It is sometimes useful to have available a binary transmission path within TELNET without having to utilize one of the more efficient, higher level protocols providing binary transmission (such as the File Transfer Protocol). The use of the IAC prefix within the basic TELNET protocol provides the option of binary transmission in a natural way, requiring only the addition of a mechanism by which the parties involved can agree to INTERPRET the characters transmitted over a TELNET connection as binary data.

50. Description of the option. With the binary transmission option in effect, the receiver should interpret characters received from the transmitter which are not preceded with IAC as 8 bit binary data, with the exception of IAC followed by IAC which stands for the 8 bit binary data with the decimal value 255. IAC followed by an effective TELNET command (plus any additional characters required to complete the command) is still the command even with the binary transmission option in effect. IAC followed by a character which is not a defined TELNET command has the same meaning as IAC followed by NOP, although an IAC followed by an undefined command should not normally be sent in this mode.

60. Implementation suggestions. It is foreseen that implementations of the binary transmission option will choose to refuse some other options (such as the EBCDIC transmission option) while the binary transmission option is in effect. However, if a pair of hosts can understand being in binary transmission mode simultaneous with being in, for example, echo mode, then it is alright if they negotiate that combination. The meanings of WON'T and DON'T are dependent upon whether the connection is presently being operated in binary mode or not. Consider a connection operating in EBCDIC mode which involves a system which has chosen not to implement any knowledge of the binary command. If this system were to receive a DO TRANSMIT-BINARY, it would not recognize the TRANSMIT-BINARY option and therefore would return a WON'T TRANSMIT-BINARY. If the default for the WON'T TRANSMIT-BINARY were always NVT ASCII, the sender of the DO TRANSMIT-BINARY would expect the recipient to have switched to NVT ASCII, whereas the receiver of the DO TRANSMIT-BINARY would not make this interpretation. Thus, we have the rule that when a connection is not presently operating in binary mode, the default (i.e., the interpretation of WON'T and DON'T) is to continue operating in the current mode, whether that is NVT ASCII, EBCDIC, or some other mode. This rule, however, is not applied once a connection is operating in a binary mode (as agreed to by both ends); this would require each end of the connection to maintain a stack, containing all of the encoding-method transitions which had previously occurred on the connection, in order to properly interpret a WON'T or DON'T. Thus, a WON'T or DON'T received after the connection is operating in binary mode causes the encoding method to revert to NVT ASCII.

MIL-STD-1782

10 May 1984

70. Binary transmission mode. It should be remembered that a TELNET connection is a two way communication channel. The binary transmission mode must be negotiated separately for each direction of data flow, if that is desired. Implementation of the binary transmission option, as is the case with implementations of all other TELNET options, must follow the loop preventing rules given in the General Considerations section of the TELNET Protocol Specification. Consider now some issues of binary transmission both to and from both a process and a terminal:

70.1 Binary transmission from a terminal. The implementer of the binary transmission option should consider how (or whether) a terminal transmitting over a TELNET connection with binary transmission in effect is allowed to generate all eight bit characters, ignoring parity considerations, etc., on input from the terminal.

70.2 Binary transmission to a process. The implementer of the binary transmission option should consider how (or whether) all characters are passed to a process receiving over a connection with binary transmission in effect. As an example of the possible problem, TOPS-20 intercepts certain characters (e.g., ETX, the terminal control-C) at monitor level and does not pass them to the process.

70.3 Binary transmission from a process. The implementer of the binary transmission option should consider how (or whether) a process transmitting over a connection with binary transmission in effect is allowed to send all eight bit characters with no characters intercepted by the monitor and changed to other characters. An example of such a conversion may be found in the TOPS-20 system where certain non-printing characters are normally converted to a Circumflex (up-arrow) followed by a printing character.

70.4 Binary transmission to a terminal. The implementer of the binary transmission option should consider how (or whether) all characters received over a connection with binary transmission in effect are sent to a local terminal. At issue may be the addition of timing characters normally inserted locally, parity calculations, and any normal code conversion.

MIL-STD-1782
10 May 1984

APPENDIX B

TELNET ECHO OPTION

10. Command name and code.

ECHO 1

20. Command meanings.

20.1 IAC WILL ECHO. The sender of this command REQUESTS to begin, or confirms that it will now begin, echoing data characters it receives over the TELNET connection back to the sender of the data characters.

20.2 IAC WON'T ECHO. The sender of this command DEMANDS to stop, or refuses to start, echoing the data characters it receives over the TELNET connection back to the sender of the data characters.

20.3 IAC DO ECHO. The sender of this command REQUESTS that the receiver of this command begin echoing, or confirms that the receiver of this command is expected to echo, data characters it receives over the TELNET connection back to the sender.

20.4 IAC DON'T ECHO. The sender of this command DEMANDS the receiver of this command stop, or not start, echoing data characters it receives over the TELNET connection.

30. Default.

WON'T ECHO

DON'T ECHO

No echoing is done over the TELNET connection.

40. Motivation for the option. The NVT has a printer and a keyboard which are nominally interconnected so that "echoes" need never traverse the network; that is to say, the NVT nominally operates in a mode where characters typed on the keyboard are (by some means) locally turned around and printed on the printer. In highly interactive situations it is appropriate for the remote process (command language interpreter, etc.) to which the characters are being sent to control the way they are echoed on the printer. In order to support such interactive situations, it is necessary that there be a TELNET option to allow the parties at the two ends of the TELNET connection to agree that characters typed on an NVT keyboard are to be echoed by the party at the other end of the TELNET connection.

MIL-STD-1782

10 May 1984

5.0 Description of the option. When the echoing option is in effect, the party at the end performing the echoing is expected to transmit (echo) data characters it receives back to the sender of the data characters. The option does not require that the characters echoed be exactly the characters received (for example, a number of systems echo the ASCII ESC character with something other than the ESC character). When the echoing option is not in effect, the receiver of data characters should not echo them back to the sender; this, of course, does not prevent the receiver from responding to data characters received. The normal TELNET connection is two way. That is, data flows in each direction on the connection independently; and neither, either, or both directions may be operating simultaneously in echo mode. There are five reasonable modes of operation for echoing on a connection pair:

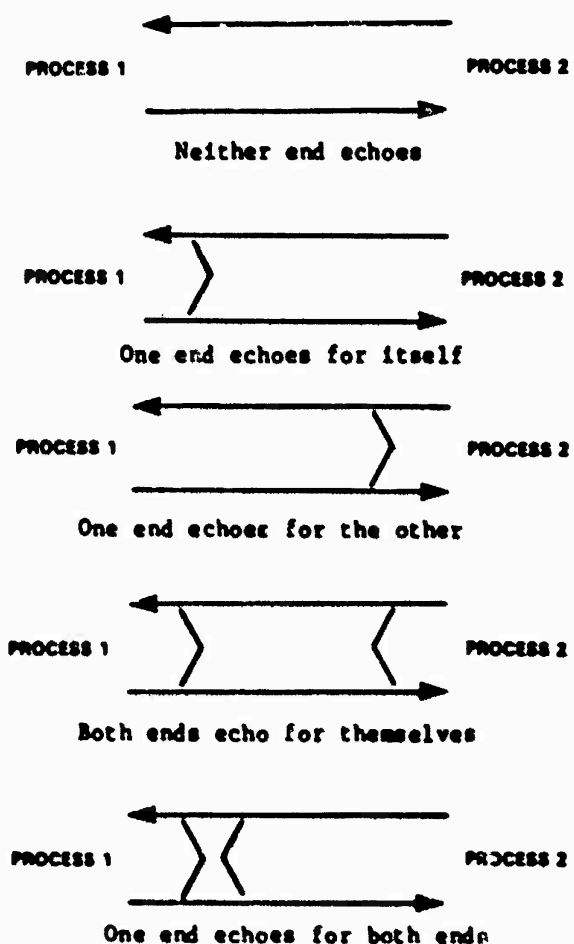


FIGURE 1. Five reasonable modes of operation for echoing on a connection pair.

MIL-STD-1782

10 May 1984

This option provides the capability to decide on whether or not either end will echo for the other. It does not, however, provide any control over whether or not an end echoes for itself; this decision must be left to the sole discretion of the systems at each end (although they may use information regarding the state of "remote" echoing negotiations in making this decision). If BOTH hosts enter the mode of echoing characters transmitted by the other host, then any character transmitted in either direction will be "echoed" back and forth indefinitely. Therefore, care should be taken in each implementation that if one site is echoing, echoing is not permitted to be turned on at the other. As discussed in Section 4, both parties to a full-duplex TELNET connection initially assume each direction of the connection is being operated in the default mode which is non-echo (non-echo is not using this option, and the same as DON'T ECHO, WON'T ECHO). If either party desires himself to echo characters to the other party or for the other party to echo characters to him, that party gives the appropriate command (WILL ECHO or DO ECHO) and waits (and hopes) for acceptance of the option. If the request to operate the connection in echo mode is refused, then the connection continues to operate in non-echo mode. If the request to operate the connection in echo mode is accepted, the connection is operated in echo mode. After a connection has been changed to echo mode, either party may demand that it revert to non-echo mode by giving the appropriate DON'T ECHO or WON'T ECHO command (which the other party must confirm thereby allowing the connection to operate in non-echo mode). Just as each direction of the TELNET connection may be put in remote echoing mode independently, each direction of the TELNET connection must be removed from remote echoing mode separately.

60. Implementation of the option. Implementations of the echo option, as implementations of all other TELNET options, must follow the loop preventing rules given in the General Requirements section of the TELNET Protocol Standard. Also, so that switches between echo and non-echo mode can be made with minimal confusion (momentary double echoing, etc.), switches in mode of operation should be made at times precisely coordinated with the reception and transmission of echo requests and demands. For instance, if one party responds to a DO ECHO with a WILL ECHO, all data characters received after the DO ECHO should be echoed and the WILL ECHO should immediately precede the first of the echoed characters. The echoing option alone will normally not be sufficient to effect what is commonly understood to be remote computer echoing of characters typed on a terminal keyboard--the SUPPRESS-GO AHEAD option will normally have to be invoked in conjunction with the ECHO option to effect character-at-a-time remote echoing.

60.1 A sample implementation of the option. The following is a description of a possible implementation for a simple user system called "UHOST". A possible implementation could be that for each user terminal, the UHOST would keep three state bits: whether the terminal echoes for itself (UHOST ECHO always) or not (ECHO mode possible), whether the (human) user prefers to operate in ECHO mode or in non-ECHO mode, and whether the connection from this terminal to the server is in ECHO or non-ECHO mode. We will call these three bits P(hysical), D(esired), and A(ctual). When a terminal dials up the UHOST the P-bit is set appropriately, the D-bit is set equal to it, and the A-bit is set to non-ECHO. The P-bit and D-bit may be manually reset by

MIL-STD-1782

10 May 1984

direct commands if the user so desires. For example, a user in Hawaii on a "full-duplex" terminal, would choose not to operate in ECHO mode, regardless of the preference of a mainland server. He should direct the UHOST to change his D-bit from ECHO to non-ECHO. When a connection is opened from the UHOST terminal to a server, the UHOST would send the server a DO ECHO command if the MIN (with non-ECHO less than ECHO) of the P- and D-bits is different from the A-bit. If a WON'T ECHO or WILL ECHO arrives from the server, the UHOST will set the A-bit to the MIN of the received request, the P-bit, and the D-bit. If this changes the state of the A-bit, the UHOST will send off the appropriate acknowledgment; if it does not, then the UHOST will send off the appropriate refusal if not changing meant that it had to deny the request (i.e., the MIN of the P-and D-bits was less than the received A-request). If while a connection is open, the UHOST terminal user changes either the P-bit or D-bit, the UHOST will repeat the above tests and send off a DO ECHO or DON'T ECHO, if necessary. When the connection is closed, the UHOST would reset the A-bit to indicate UHOST echoing. While the UHOST's implementation would not involve DO ECHO or DON'T ECHO commands being sent to the server except when the connection is opened or the user explicitly changes his echoing mode, bigger hosts might invoke such mode switches quite frequently. For instance, while a line-at-a-time system were running, the server might attempt to put the user in local echo mode by sending the WON'T ECHO command to the user; but while a character-at-a-time system were running, the server might attempt to invoke remote echoing for the user by sending the WILL ECHO command to the user. Furthermore, while the UHOST will never send a WILL ECHO command and will only send a WON'T ECHO to refuse a server sent DO ECHO command, a server host might often send the WILL and WON'T ECHO commands.

MIL-STD-1782
10 May 1984

APPENDIX C

TELNET SUPPRESS GO AHEAD OPTION

10. Command name and code.

SUPPRESS-GO-AHEAD 3

20. Command meanings.

20.1 IAC WILL SUPPRESS-GO-AHEAD. The sender of this command requests permission to begin suppressing transmission of the TELNET GO AHEAD (GA) character when transmitting data characters, or the sender of this command confirms it will now begin suppressing transmission of GAs with transmitted data characters.

20.2 IAC WON'T SUPPRESS-GO-AHEAD. The sender of this command demands to begin transmitting, or to continue transmitting, the GA character when transmitting data characters.

20.3 IAC DO SUPPRESS-GO-AHEAD. The sender of this command requests that the sender of data start suppressing GA when transmitting data, or the sender of this command confirms that the sender of data is expected to suppress transmission of GAs.

20.4 IAC DON'T SUPPRESS-GO-AHEAD. The sender of this command demands that the receiver of the command start or continue transmitting GAs when transmitting data.

30. Default.

WON'T SUPPRESS-GO-AHEAD

DON'T SUPPRESS-GO-AHEAD

Go aheads are transmitted.

40. Motivation for the option. While the NVT nominally follows a half duplex protocol complete with a GO AHEAD signal, there is no reason why a full duplex connection between a full duplex terminal and a host optimized to handle full duplex terminals should be burdened with the GO AHEAD signal. Therefore, it is desirable to have a TELNET option with which parties involved can agree that one or the other or both should suppress transmission of GO AHEADS.

50. Description of the option. When the SUPPRESS-GO-AHEAD option is in effect on the connection between a sender of data and the receiver of the data, the sender need not transmit GAs. It seems probable that the parties to the TELNET connection will suppress GO AHEAD in both directions of the TELNET connection if GO AHEAD is suppressed at all; but, nonetheless, it

MIL-STD-1782

10 May 1984

must be suppressed in both directions independently. With the SUPPRESS-GO-AHEAD option in effect, the IAC GA command should be treated as a NOP if received, although IAC GA should not normally be sent in this mode.

60. Implementation considerations. As the SUPPRESS-GO-AHEAD option is sort of the opposite of a line-at-a time mode, the sender of data which is suppressing GO AHEADs should attempt to actually transmit characters as soon as possible (i.e., with minimal buffering) consistent with any other agreements which are in effect. In many TELNET implementations it will be desirable to couple the SUPPRESS-GO-AHEAD option to the echo option so that when the echo option is in effect, the SUPPRESS-GO-AHEAD option is in effect simultaneously: both of these options will normally have to be in effect simultaneously to effect what is commonly understood to be character-at-a time echoing by the remote computer.

MIL-STD-1782
10 May 1984

APPENDIX D

TELNET STATUS OPTION

10. Command name and code.

STATUS 5

20.1 Command meanings. This option applies separately to each direction of data flow.

20.1 IAC WILL STATUS. The sender of WILL status agrees to send status information, spontaneously or in response to future requests.

20.2 IAC WON'T STATUS. Sender refuses to carry on any further discussion of the current status of options.

20.3 IAC DO STATUS. The sender of DO wishes to be able to send request for status of in option information or confirms that he is willing to send such requests.

20.4 IAC DON'T STATUS. Sender refuses to carry on any further discussion of the current status of options.

20.5 IAC SB STATUS SEND IAC SE. Sender requests receiver to transmit his (the receiver's) perception of the current status of TELNET options. The code for SEND is 1.

20.6 IAC SB STATUSIS...IAC SE. Sender is stating his perception of the current status of TELNET options. The code for IS is 0.

30. Default.

DON'T STATUS, WON'T STATUS

The current status of options will not be discussed.

40. Motivation for the option. This option allows a user/process to verify the current status of TELNET options (e.g., echoing) as viewed by the person/process on the other end of the TELNET connection. Simply renegotiating options could lead to the nonterminating request loop problem discussed in paragraph 4.2.3. This option fits into the normal structure of TELNET options by deferring the actual transfer of status information to the SB command.

50. Description of the option. WILL and DO are used only to obtain and grant permission for future discussion. The actual exchange of status information occurs within option subcommands (IAC SB STATUS...). Once the two hosts have exchanged a WILL and a DO, the sender of the WILL STATUS is free to transmit status information, spontaneously or in response to a request from

MIL-STD-1782

10 May 1984

the sender of the DO. At worst, this may lead to transmitting the information twice. Only the sender of the DO may send requests (IAC SB STATUS SEND IAC SE) and only the sender of the WILL may transmit actual status information (within an IAC SB STATUS IS ... IAC SE command). IS has the subcommands WILL, DO and SB. They are used EXACTLY as used during the actual negotiation of TELNET options, except that SB is terminated with SE, rather than IAC SE. Transmission of SE, as a regular data byte, is accomplished by doubling the byte (SE SE). Options that are not explicitly described are assumed to be in their default states. A single IAC SB STATUS IS...IAC SE describes the condition of ALL options.

60. Example of option application. The following is an example of use of the option:

Host1: IAC DO STATUS

Host2: IAC WILL STATUS

(Host2 is now free to send status information at any time.
Solicitations from Host1 are NOT necessary. This should not produce
any dangerous race conditions. At worst, two IS's will be sent.)

Host1 (perhaps): IAC SB STATUS SEND IAC SE

Host2 (the following stream is broken into multiple lines only for
readability. No carriage returns are implied.):

IAC SB STATUS IS

WILL ECHO

DO SUPPRESS-GO-AHEAD

WILL STATUS

DO STATUS

IAC SE

Explanation of Host2's perceptions: It is responsible for echoing
back the data characters it receives over the TELNET connection;
it will not send Go-Ahead signals; it will both issue and request
Status information.

MIL-STD-1782
10 May 1984

APPENIX E

TELNET TIMING MARK OPTION

1C. Command name and code.

TIMING-MARK 6

20. Command meanings.

20.1 IAC WILL TIMING-MARK. The sender of this command ASSURES the receiver of this command that it is inserted in the data stream at the "appropriate place" to insure synchronization with a DO TIMING-MARK transmitted by the receiver of this command.

20.2 IAC WON'T TIMING-MARK. The sender of this command REFUSES to insure that this command is inserted in the data stream at the "appropriate place" to insure synchronization.

20.3 IAC DO TIMING-MARK. The sender of this command REQUESTS that the receiver of this command return a WILL TIMING-MARK in the data stream at the "appropriate place" as defined in paragraph 11.4 below.

20.4 IAC DON'T TIMING-MARK. The sender of this command notifies the receiver of this command that a WILL TIMING-MARK (previously transmitted by the receiver of this command) has been IGNORED.

30. Default.

WON'T TIMING-MARK, DON'T TIMING-MARK

i.e., No explicit attempt is made to synchronize the activities at the two ends of the TELNET connection.

40. Motivation for the option. It is sometimes useful for a user or process at one end of a TELNET connection to be sure that previously transmitted data has been completely processed, printed, discarded, or otherwise disposed of. This option provides a mechanism for doing this. In addition, even if the option request (DO TIMING-MARK) is refused (by WON'T TIMING-MARK) the requester is at least assured that the refuser has received (if not processed) all previous data.

50. Examples of option application. As an example of a particular application, imagine a TELNET connection between a physically full duplex terminal and a "full duplex" server system which permits the user to "type ahead" while the server is processing previous user input. Suppose that both sides have agreed to Suppress Go Ahead and that the server has agreed to provide echoes. The server now discovers a command which it cannot parse, perhaps because of a user typing error. It would like to throw away all of the user's "type-ahead" (since failure of the parsing of one command is likely

MIL-STD-1782

10 May 1984

to lead to incorrect results if subsequent commands are executed), send the user an error message, and resume interpretation of commands which the user typed after seeing the error message. If the user were local, the system would be able to discard the buffered input; but input may be buffered in the user's host or elsewhere. Therefore, the server might send a DO TIMING-MARK and hope to receive a WILL TIMING-MARK from the user at the "appropriate place" in the data stream. The "appropriate place" (in absence of other information) is clearly just before the first character which the user typed after seeing the error message. That is, it should appear that the timing mark was "printed" on the user's terminal and that, in response, the user typed an answering timing mark. Next, suppose that the user in the example above realized that he had misspelled a command, realized that the server would send a DO TIMING-MARK, and wanted to start "typing ahead" again without waiting for this to occur. He might then instruct his own system to send a WILL TIMING-MARK to the server and then begin "typing ahead" again. (Implementers should remember that the user's own system must remember that it sent the WILL TIMING-MARK so as to discard the DO/DON'T TIMING-MARK when it eventually arrives.) Thus, in this case the "appropriate place" for the insertion of the WILL TIMING-MARK is the place defined by the user. In both of the examples above, it is the responsibility of the system which transmits the DO TIMING-MARK to discard any unwanted characters; the WILL TIMING-MARK only provides help in deciding which characters are "unwanted".

6.0 Description of the option. Suppose that Process A of Figure 2 wishes to synchronize with B. The DO TIMING-MARK is sent from A to B. B can refuse by replying WON'T TIMING-MARK, or agree by permitting the timing mark to flow through his "outgoing" buffer, BUF2. Then, instead of delivering it to the terminal, B will enter the mark into his "incoming" buffer BUF1, to flow through toward A. When the mark has propagated through B's incoming buffer, B returns the WILL TIMING-MARK over the TELNET connection to A.

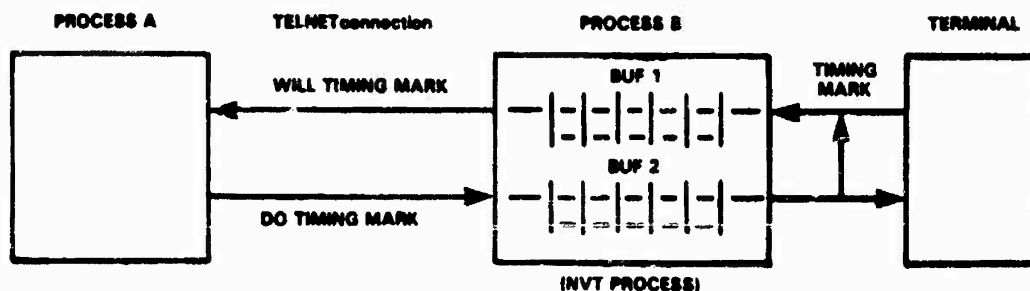


FIGURE 2. Synchronization of processes.

When A receives the WILL TIMING-MARK, he knows that all the information he sent to B before sending the timing mark been delivered, and all the information sent from B to A before turnaround of the timing mark has been delivered.

MIL-STD-1782
10 May 1984

70. Three typical applications.

70.1 Round-trip delay. Measure round-trip delay between a process and a terminal or another process.

70.2 Resynchronization. Resynchronizing an interaction is described in paragraph 4 above. A is a process interpreting commands forwarded from a terminal by B. When A sees an illegal command it:

- a. Sends <carriage return>, <line feed>, <question mark>.
- b. Sends DO TIMING-MARK.
- c. Sends an error message.
- d. Starts reading input and throwing it away until it receives a WILL TIMING-MARK.
- e. Resumes interpretation of input.

This achieves the effect of flushing all "type ahead" after the erroneous command, up to the point when the user actually saw the question mark.

70.3 Dual synchronization. The dual of paragraph 7.2. The terminal user wants to throw away unwanted output from A.

- a. B sends DO TIMING-MARK, followed by some new command.
- b. B starts reading output from A and throwing it away until it receives WILL TIMING-MARK.
- c. B resumes forwarding A's output to the terminal.

This achieves the effect of flushing all output from A, up to the point where A saw the timing mark, but not output generated in response to the following command.

MIL-STD-1782
10 May 1984

APPENDIX F

TELNET EXTENDED OPTIONS - LIST OPTION

10. Command name and code.

EXTENDED-OPTIONS-LIST (EXOPL) 255

20. Command meanings.

20.1 IAC WILL EXOPL. The sender of this command REQUESTS permission to begin negotiating, or confirms that it will begin negotiating, TELNET options which are on the "Extended Options List."

20.2 IAC WON'T EXOPL. The sender of this command REFUSES to negotiate, or to continue negotiating, options on the "Extended Options List."

20.3 IAC DO EXOPL. The sender of this command REQUESTS that the receiver of this command begin negotiating, or confirms that the receiver of this command is expected to begin negotiating, TELNET options which are on the "Extended Options List."

20.4 IAC DON'T EXOPL. The sender of this command DEMANDS that the receiver conduct no further negotiation of options on the "Extended Options List."

20.5 IAC SB EXOPL <subcommand>. The subcommand contains information required for the negotiation of an option of the "Extended Options List." The format of the subcommand is discussed in paragraph 5 below.

30. Default.

WON'T EXOPL, DON'T EXOPL

Negotiation of options on the "Extended Options List" is not permitted.

40. Motivation for the option. Eventually, a 257th TELNET option will be needed. This option will extend the option list for another 256 options in a manner which is easy to implement. The option is proposed now, rather than later (probably much later), in order to reserve the option number (255).

50. Description of the option. The EXOPL option has five subcommand codes: WILL, WON'T, DO, DON'T, and SB. They have exactly the same meanings as the TELNET commands with the same names, and are used in exactly the same way. For consistency, these subcommand codes will have the same values as the TELNET command codes (250-254). Thus, the format for negotiating a specific option on the "Extended Options List" (once both parties have agreed to use it) is:

IAC SB EXOPL DO/DON'T/WILL/WON'T/<option code> IAC SE

MIL-STD-1782
10 May 1984

Once both sides have agreed to use the specific option specified by <option code>, subnegotiation may be required. In this case the format to be used is:

IAC SB EXOPL SB <option code> <parameters> SE IAC SE

* U.S. GOVERNMENT PRINTING OFFICE: 1984-705-040: A3855



DEFENSE COMMUNICATIONS AGENCY

DEFENSE DATA NETWORK X.25 HOST INTERFACE SPECIFICATION

DECEMBER 1983

PREPARED BY BBN COMMUNICATIONS CORPORATION

DDN

ACKNOWLEDGEMENTS

This specification was prepared by BBN Communications Corporation under contract to the Defense Data Network Program Management Office of the Defense Communications Agency.

The specification has been reviewed by the Defense Communications Engineering Center for accuracy and completeness. The draft of this specification has been disseminated to industry by the National Bureau of Standards for review and comments which have been incorporated in the final specification. This specification has been approved for use on the Defense Data Network by the DoD Protocol Standards Steering Group.

Comments on this specification should be directed to the Defense Communications Agency, ATTN: Defense Data Network Program Management Office, Code B610, Washington, D.C. 20305

Table of Contents

1	INTRODUCTION.....	1
1.1	Background.....	1
1.1.1	X.25 and FIPS 100/Federal Standard 1041.....	1
1.1.2	X.25-to-X.25 and X.25-to-1822 Interoperability.....	2
1.2	Compliance.....	4
1.2.1	Compliance With CCITT X.25 and FIPS 100/Fed. Std. 1041.....	4
1.2.2	DTE Compliance With This Specification.....	4
2	INTERFACE SPECIFICATION.....	6
2.1	Call Establishment Conventions.....	6
2.1.1	Addressing.....	6
2.1.1.1	Address Formats and Fields.....	6
2.1.1.1.1	Reserved.....	7
2.1.1.1.2	Flag.....	7
2.1.1.1.3	DDN Host Identifier.....	7
2.1.1.1.4	Sub-Address.....	7
2.1.1.2	Supplying Missing Address Information.....	7
2.1.2	DDN-Specific Facilities.....	8
2.1.2.1	Type of Service Selection.....	8
2.1.2.2	Call Precedence.....	9
2.1.3	Protocol Identification.....	10
2.1.4	Logical Channel Assignment.....	10
2.2	Packet Level Procedures.....	11
2.3	Link Level Procedures.....	12
2.3.1	Link Level Parameters and Options.....	12
2.3.2	Timer T1 and Parameter T2.....	12
2.3.3	Maximum I Frame Size.....	13
2.4	Physical Level Specifications.....	14
3	BIBLIOGRAPHY.....	16
	APPENDIX A: DDN X.25 Implementation Details.....	A-1
A-1	Introduction.....	A-1
A-2	Operational Features of DDN X.25 DCE Releases.....	A-1
A-2.1	Initial Feature Support.....	A-1
A-2.2	Exception-Handling Procedures.....	A-2
A-2.2.1	Non-Octet-Aligned Data.....	A-2
A-2.2.2	RESTART REQUEST Packet.....	A-2
A-2.2.3	RESET REQUEST Packet.....	A-2
A-2.2.4	CLEAR REQUEST Packet.....	A-3
A-2.3	Virtual Circuit Resource Availability.....	A-3

A-3	Detailed Features and Facilities	
	Specifications.....	A-3
A-3.1	Additional Diagnostic Codes.....	A-3
A-3.2	X.25 IP Interoperability Considerations.....	A-6
A-3.3	The DDN Logical Addressing Facility.....	A-7
A-3.3.1	Logical Addresses.....	A-7
A-3.3.2	Enabling and Disabling Logical Addresses.....	A-7
A-4	Limitations of DDN Basic X.25 Service.....	A-8
A-5	Derivation of DDN X.25 Addresses.....	A-9
APPENDIX B: DDN Synchronous Level 1 Specification.....		B-1
B-1	Introduction.....	B-1
B-2	Supported Interfaces.....	B-1
APPENDIX C: Federal Information Processing Standard		
	Publication 100.....	C-1

TABLES

DDN X.25 Address Fields.....	7
Derivation of Maximum I Frame Size.....	14
DDN X.25 Physical Signaling Rates and Interfaces.....	15
Additional Packet Level Diagnostic Codes.....	A-4
IP Precedence to X.25 Precedence Mapping.....	A-6
EIA and CCITT Interchange Circuits.....	B-3
Signal Selection by CCITT Interchange Circuit Number.....	B-4
Typical Level 1 Connection Schemes.....	B-5
Interface Type by Service Speed.....	7
RS-232-C Interface.....	B-8
MIL-188-114 Interface (and equivalents).....	B-9
V.35 Interface.....	B-10

FIGURES

Typical Level 1 Connection Schemes..... B-4

1 INTRODUCTION

This report specifies the attachment of an X.25 host to the Defense Data Network (DDN). In particular, this report describes specific options and features of CCITT Recommendation X.25 (1980) and Federal Information Processing Standard (FIPS) 100/Federal Standard (Fed. Std.) 1041 (July 1983) required of a host X.25 implementation to enable that host to communicate with a DDN X.25 Interface Message Processor ("IMP", the DDN packet switching node). This report, in conjunction with FIPS 100/Fed. Std. 1041, should enable DDN host site managers and others planning to attach a host by means of X.25, rather than the 1822 interface, * to determine, first, whether or not the X.25 implementation of the host in question is adequate for operation with DDN, and, second, what options, parameter settings, etc. must or may be selected for operation with DDN.

This report assumes that the reader is familiar with CCITT Recommendation X.25 and FIPS 100/Fed. Std. 1041. A copy of FIPS 100/Fed. Std. 1041 is attached as Appendix C of this report.

In this document, the term "Administration" refers to the Defense Communications Agency (DCA Code B610, Washington, D. C. 20305).

1.1 Background

1.1.1 X.25 and FIPS 100/Federal Standard 1041

The CCITT Recommendation X.25 describes the interface between host computers (data terminal equipment, or DTEs) and data circuit-terminating equipment (DCEs, which effect communication with remote hosts over computer networks) for hosts operating in the packet mode on public data networks. The X.25 interface standard is defined as three independent architectural levels, following the Open Systems Interconnection (OSI) Reference Model. The three levels are:

Level 1: The PHYSICAL level of the connection. The physical, electrical, functional, and procedural characteristics to activate,

* As used in this report, "1822 interface" refers to the interface specified in Bolt Beranek and Newman Inc. (BBN) Report No. 1822, "Specification for the Interconnection of a Host and an IMP," revision of December 1981.

maintain, and deactivate the physical link between the DTE and the DCE.

Level 2: The LINK level of the connection. The link access procedure for data interchange across the link between the DTE and the DCE.

Level 3: The PACKET level of the connection. The packet format and control procedures for the exchange of packets containing control information and user data between the DTE and the DCE, and between the DTE and a remote DTE.

CCITT Recommendation X.25 contains many options and implementation choices. FIPS 100/Fed. Std. 1041, which specifies the general use of X.25 for the Federal Government, defines some of the choices left open in X.25. This document describes the X.25 interface to a particular network, DDN. Thus in several areas where X.25 allows a choice, a single choice appropriate for DDN is specified; in areas which X.25 leaves unspecified, addressing in particular, conventions are specified that are consistent with the overall architecture of DDN and the interoperability goals described below. The effect of this approach is to make DDN service available to hosts in a way that requires no changes to a host DTE implementation that is compliant with FIPS 100/Fed. Std. 1041 and CCITT Recommendation X.25. By implementing extensions described in this specification, a host will be able to take advantage of additional DDN features required in military networks, such as precedence and logical addressing.

The reader is referred to CCITT Recommendation X.25 and to FIPS 100/Fed. Std. 1041 for detailed information not provided in the body of this document.

1.1.2 X.25-to-X.25 and X.25-to-1822 Interoperability

A key goal of the DDN X.25 implementation is interoperability among all DDN subscribers. That is, effective communication should be possible, not only between subscribers attached to the DDN using identical vendor-supplied X.25 implementations, but between subscribers using different X.25 implementations, and between a subscriber using an X.25 interface to the DDN and a subscriber using an 1822 interface to the DDN. Achieving this goal of interoperability requires that all DDN

X.25 subscribers conform to this interface specification and implement the DoD standard higher level protocols. True interoperability among DDN hosts requires, in particular, implementation of the DoD standard protocols TCP (Transmission Control Protocol) and IP (Internet Protocol), as well as the higher-level protocols which implement DDN standard services, when such services are provided by the host: the Telnet Protocol for character-oriented terminal support, the File Transfer Protocol (FTP) for file movement between hosts, and the Simple Mail Transfer Protocol (SMTP) for communication between electronic mail service hosts.

The DDN X.25 DCE offers two types of service to X.25 DTEs:

1. DDN Standard X.25 Service, which, when used in conjunction with DoD standard protocols, provides interoperable communication between an X.25 DTE and other DDN hosts that also implement the DoD standard protocols, whether they are connected to DDN via the 1822 interface or via the X.25 interface;

and

2. DDN Basic X.25 Service, which provides communication only between an X.25 DTE and other DDN X.25 DTEs implementing compatible higher-level protocols.

Section 2.1.2.1 of this report describes the conventions to be used by a DTE to specify the type of service desired for each X.25 virtual call. All DDN X.25 DTEs will be required to develop and initiate a plan to use the DoD standard protocol architecture and DDN standard X.25 service.

Use of DDN basic X.25 service imposes some restrictions on the nature of the network communications service that a host can obtain. These restrictions are discussed in Appendix A, Section A-4.

1.2 Compliance

1.2.1 Compliance With CCITT X.25 and FIPS 100/Fed. Std. 1041

The DDN X.25 Interface Specification is compliant with CCITT Recommendation X.25 and FIPS 100/Fed. Std. 1041. The DDN X.25 DCE supports all facilities specified as E (essential) by FIPS 100/Fed. Std. 1041, and most facilities specified as A (additional). The additional facilities not supported are:

- (i) datagrams and associated facilities,
and
- (ii) bilateral closed user groups.

In that FIPS 100/Fed. Std. 1041 describes features for a DCE, DDN X.25 DTEs may support any or all facilities specified as either E or A by FIPS 100/Fed. Std. 1041. However, DDN X.25 DTEs must not use the facilities identified above that are not supported by the DDN X.25 DCE.

1.2.2 DTE Compliance With This Specification

This document specifies several areas in which the DDN X.25 DCE is capable of operating in several modes. For example, Section 2.4 lists a number of signaling rates supported by the DCE. In such cases, a DDN X.25 DTE must implement at least one of the options listed (or the set of options required of a DTE by FIPS 100/Fed. Std. 1041) but need not implement all of the options listed (unless required by FIPS 100/Fed. Std. 1041). Determining the adequacy of the options supported by a DTE vendor for meeting a DDN subscriber's requirements is the responsibility of the subscriber.

In addition to the CCITT X.25 and FIPS 100/Fed. Std. 1041 requirements described in Section 1.2.1 above, DDN X.25 DTEs may wish to take advantage of additional DDN-specific features that are compatible extensions to the public standards. Implementation of a DDN-specific feature by a host is required only if the host wishes to take advantage of the service or information provided by the feature. For example, a host that wishes to establish calls only at the default precedence level assigned to it need not implement the precedence facility described in Section 2.1.2.2. However, a host that wishes to have flexibility in the precedence of the calls it establishes must implement this facility.

Any deficiencies with respect to this specification in a vendor-supplied X.25 DTE implementation contemplated for use with the DDN X.25 DCE should be rectified so as to attain compliance with this specification. Proper operation with DDN of an X.25 DTE that is not compliant with this specification cannot be guaranteed and should not be attempted. To this end, a test program is available through the Administration.

2 INTERFACE SPECIFICATION

2.1 Call Establishment Conventions

This section specifies DDN X.25 call establishment conventions.

2.1.1 Addressing

DDN addresses are assigned to subscriber DTEs by the Administration. Two basic forms of address are provided: physical addresses, which correspond to the node number and DCE port number of the node to which the DTE is connected, and logical addresses, which are mapped transparently by DCE software into a corresponding physical network address. Each DTE is assigned one physical address, and may be assigned one or more logical addresses. All DDN addresses are either twelve or fourteen BCD (binary-coded decimal) digits in length. A calling DTE need not determine whether a given address is a physical or logical address, in order to establish a call to that address.

2.1.1.1 Address Formats and Fields

DDN addresses have the following format:

ZZZZ F DDDDDDD (SS)

The various fields of the address are presented in Table 2.1 and are explained below.

Field	Meaning	Length (BCD digits)
ZZZZ	Reserved (must be zero)	4
F	Flag	1
DDDDDDD	DDN Host Identifier	7
(SS)	Sub-address (optional)	0 or 2
TOTAL		12 or 14

Table 2.1 DDN X.25 Address Fields

DTE implementors are cautioned that use of this mechanism in accepting calls to a DTE's logical address (See Appendix A, Section A-3.3) can result in confusion on the part of the calling DTE and is not advised.

2.1.2 DDN-Specific Facilities

Two DDN-specific features are requested by means of "private" or non-CCITT facilities in CALL REQUEST and CALL ACCEPTED packets. If either or both of these facilities are requested in a CALL REQUEST or CALL ACCEPTED packet, they must follow all CCITT X.25 facilities and must be preceded by a single facility marker, two octets of zero.

2.1.2.1 Type of Service Selection

The DDN X.25 provides two types of service, DDN basic X.25 service and DDN standard X.25 service. DDN standard X.25 service provides only local DTE to local DCE support of the X.25 connection. Data is carried via the network to its destination (using protocols internal to the network), where it is delivered using the access protocol of the destination host (i.e., either 1822 or DDN standard X.25 service). This access method is oriented towards DDN X.25 hosts using the DoD standard TCP/IP higher level protocols. No X.25 procedures change when using DDN standard X.25 service; however, the significance of the procedures changes (see Appendix A, Section A-3.2). There is no end-to-end X.25-level acknowledgement or guarantee of delivery of data packets with DDN standard X.25 service; reliability of DDN standard X.25 service is provided instead by the use of a reliable transport protocol.

DDN basic X.25 service provides end-to-end call management with significance as described in CCITT Recommendation X.25 and FIPS 100/Fed. Std. 1041. This access method is oriented towards hosts that have existing higher level protocol implementations that require reliable packet delivery at the network level.

Selection of DDN standard or DDN basic X.25 service must be made on a call-by-call basis by the DDN X.25 DTE at the time of call setup. To specify DDN standard X.25 service, a DTE must include in the CALL REQUEST packet a facility two octets long, coded as follows:

00000100 00000001

2.1.1.1.1 Reserved

The Reserved field corresponds to the DNIC field generally used in public data networks. Pending assignment of a DDN DNIC, this field must be zero.

2.1.1.1.2 Flag

The Flag field is used to differentiate physical and logical addressing. The value zero indicates physical addressing, while the value one indicates logical addressing. A value of nine is used in the setup of calls to enable and disable logical addresses; see Appendix A, Section A-3.3.1.

2.1.1.1.3 DDN Host Identifier

The DDN Host Identifier is a seven-digit address, either logical or physical, assigned to a subscriber DTE by the DDN Administration.

2.1.1.1.4 Sub-Address

The Sub-Address may be used by a DTE for any purpose. It is carried across the network without modification. Its presence is optional.

2.1.1.2 Supplying Missing Address Information

The DDN X.25 DCE incorporates a mechanism to supply "missing" address information in CALL REQUEST and CALL ACCEPTED packets received from an attached DTE. This mechanism is useful in DTE software testing and physical address determination.

If a DTE sends a CALL REQUEST packet with no calling address field, the local DCE will insert the physical calling DDN Host Identifier with no subaddress field. If a DTE sends a CALL REQUEST or CALL ACCEPTED packet with either or both calling or called addresses that contain F = zero and DDDDDDD = zero, the local DCE will replace the DDN Host Identifier field (DDDDDD) with the physical address of the DTE.

If this facility is not specified, DDN basic X.25 service will be provided.

2.1.2.2 Call Precedence

The precedence of a call is negotiated by an X.25 DTE by means of a facility two octets long, coded as:

00001000 000000XX

where XX is the precedence, from 0 (lowest precedence) to 3 (highest precedence). If this facility is not used, the call will be established at the subscriber's default precedence.

A DTE is not permitted to establish a call at a precedence level higher than that authorized for that DTE by the Administration. An attempt to do so will result in the DDN X.25 DCE returning to the DTE a CLEAR INDICATION packet with clearing cause 00001001, "Out of order," with diagnostic code 194, "Requested precedence too high."

Calls of a lower precedence may be cleared by a DCE if DCE or other network resources are required, or if access to the local or remote DTE is required (for a call of higher precedence). In this event, a CLEAR INDICATION packet will be sent with the clearing cause 00000101, "Network congestion," and with a diagnostic code specifying the reason for the preemption. The diagnostic codes employed for this purpose are 192, "Cleared due to higher precedence call at local DCE," and 193, "Cleared due to higher precedence call at remote DCE." Similarly, an attempt to establish a call may be unsuccessful if network resources are engaged in calls of higher priority than that requested. In this case, a CLEAR INDICATION packet will be sent with the clearing cause 00001001, "Out of order," and with either diagnostic code 192 or 193, as appropriate.

The diagnostic codes described in the preceding paragraphs are DDN-specific diagnostic codes; additional information about these codes may be found in Appendix A, Section A-3.1.

2.1.3 Protocol Identification

X.25 DTEs employing the DoD standard TCP/IP protocol architecture must indicate this by means of the call user data field of the CALL REQUEST packet. The first octet of this field must be set to 11001100 to identify the DoD standard protocol architecture.

Indication of the use of the DoD standard protocol architecture is independent of the selection of DDN standard or DDN basic X.25 service by means of the facility specified in Section 2.1.2.1 above. Therefore, a host employing the DoD standard protocol architecture and using DDN standard X.25 service must include both the DDN standard X.25 service facility and the call user data DoD standard protocol identification in its CALL REQUEST packet.

A DTE using a protocol architecture other than the standard DoD protocol architecture is free to use any call user data protocol identification recognized by the DTEs with which it wishes to communicate. Identification of protocol architectures other than the DoD standard architecture is not standardized or enforced by the Administration. Subscribers are cautioned, therefore, that conflicts among various vendor-assigned protocol identifications may arise.

2.1.4 Logical Channel Assignment

The assignment of logical channels by the DDN X.25 DCE follows the requirements and guidelines of FIPS 100/Fed. Std. 1041 and Annex A of CCITT X.25. Within the guidelines of CCITT X.25 Annex A, the range of logical channel numbers assigned to permanent virtual circuits, incoming, two-way, and outgoing virtual calls for DDN DCEs is configured for each DTE attached to a DCE by the Administration.

DDN X.25 DTEs must follow the logical channel selection requirements of FIPS 100/Fed. Std. 1041.

The number of logical channels available to a DTE is dependent upon the configuration of the DCE to which the DTE is attached, and upon the dynamic requirements placed upon other DCEs that share the same DDN packet switching node.

2.2 Packet Level Procedures

DDN X.25 packet level procedures are as specified by FIPS 100/Fed. Std. 1041 and CCITT X.25. The following additional information is provided:

1. The maximum window size that may be negotiated is seven.
2. Modulo 128 packet level sequence numbering is not supported.
3. Maximum packet sizes of 16, 32, 64, 128, 256, 512, and 1024 octets may be negotiated.
4. The DDN X.25 DCE uses additional packet level diagnostic codes, specified in Appendix A, Table A-1. DDN X.25 DTEs may, but are not required to, make use of the information conveyed by these codes.
5. The Qualifier bit (Q-bit) is passed transparently by the DDN X.25 DCE in DDN basic X.25 service. DTEs using DDN basic X.25 service may use the Q-bit in any way that is consistent with FIPS 100/Fed. Std. 1041.
6. The DDN X.25 DCE implements the diagnostic packet. It is sent under conditions specified in Annex D of CCITT X.25. The DTE is not required to act on the information provided in diagnostic packets.
7. DTEs using DDN standard X.25 service must restrict the maximum number of data bits in a complete packet sequence to be no more than 8056. This ensures that the data from a packet sequence transmitted by an X.25 host will fit within the maximum 1822 message length limit upon delivery to an 1822 host. This restriction is necessary as existing 1822 host implementations are not required to accept messenger longer than 8063 bits.*

* DTEs using DDN standard X.25 service will generally be transmitting Internet Protocol datagrams, the length of which, by convention, does not approach this limit. Therefore, unless a protocol other than the Internet Protocol is used with DDN standard X.25 service, this is a technical restriction that will have no practical impact upon the design of DTE software. See Appendix A, Section A-3.2.

DDN X.25 DTEs connecting to DDN through an X.25 Internet Private Line Interface (IPLI) must reduce the maximum complete packet sequence length by an additional 256 bits to allow for IPLI overhead.

2.3 Link Level Procedures

DDN X.25 link level procedures are as specified by FIPS 100/Fed. Std. 1041 and CCITT X.25. This section presents additional information.

2.3.1 Link Level Parameters and Options

1. The default value of K, the maximum number of sequentially numbered I frames that the DCE will have outstanding (unacknowledged) at any given time, is seven. A DDN X.25 DCE may be configured on a per-DTE basis to provide optional values of K from one to six.
2. The default value of N2, the maximum number of transmissions and retransmissions of a frame following the expiration of the T1 timer, is twenty. This value can be changed to any value from one to 200 as a DCE configuration parameter on a per-DTE basis.
3. The optional 32-bit FCS is not supported.

2.3.2 Timer T1 and Parameter T2

The period of the timer T1 used by the DDN X.25 DCE reflects assumptions about the processing speed of the DTE. The DCE assumes that parameter T2, the response latency of the DTE to a frame from the DCE, is no greater than 1/2 second. Likewise, the DCE guarantees that its parameter T2, the latency in responding to frames from the DTE, is 1/2 second for signaling rates of 19.2 Kb/s or slower, and 1/4 second for faster links.

A lower bound for timer T1 may be computed to be $4X + T2$, based on the assumptions that:

- the link propagation time is negligible,

- * the worst-case frame transmission time is X,
- * timer T1 is started when a frame is scheduled for output,
- * each frame is scheduled just as transmission of the previous frame starts,
- * frames are not aborted, and
- * each frame and its predecessor are of maximum length N1 = 8248 bits (see Section 2.3.3 below).

As an example, for a signaling rate of 9.6 Kb/s, this yields $X = .86$ sec. If T2 is .5 sec., the total time for the DTE to respond in the worst case should be 3.9 seconds. In fact, the DCE uses a T1 timer value of 4 seconds for a link speed of 9.6 Kb/s.

In no case does the DCE use a value for T1 smaller than 3 seconds. This means that, for faster links, the DTE's T2 parameter may be lengthened because the X term in the above formula is smaller. For links of 19.2 Kb/s or faster, DTEs are expected to satisfy latency requirements that allow the DCE to use the formula $4X + T2 \text{ (DTE)} < 3 \text{ seconds} = T1 \text{ (DCE)}$.

The DTE may choose any value for T1 that is compatible with the DCE's T2 parameter values. The value of T1 used by the DTE may always be set longer than the formula indicates, with the result that recovery from certain types of link errors will be slower. However, the DCE's parameter T2 cannot be reduced, so the formula should be viewed as yielding a lower bound on the DTE's T1 timer.

2.3.3 Maximum I Frame Size

The maximum number N1 of bits in an I Frame is 8248, accommodating a data packet with up to 1024 data octets. The derivation of this number is shown in Table 2.2.

DTEs using DDN standard X.25 service must observe the restriction on the number of data bits in a complete packet sequence given in Section 2.2 above.

Field Name	X.25 Level	No. of Bits
Address	2	8
Control	2	8
General Format Identifier	3	4
Logical Channel Number	3	12
Packet Type	3	8
User Data	3	8192 (max)
Frame Check Sequence	2	16
TOTAL		8248 (max)

Table 2.2 Derivation of Maximum I Frame Size

2.4 Physical Level Specifications

The DDN X.25 physical level specification is in conformance with FIPS 100/Fed. Std. 1041 and CCITT X.25. This section presents additional information.

A DDN X.25 DTE may either be collocated with its DCE or may be connected to it via an access line. In all cases the DTE presents a physical DTE interface; the DDN will supply the matching DCE interface. DDN X.25 service offers four physical level interfaces: RS-232-C (CCITT V.28), RS-449, both balanced and unbalanced (CCITT V.11 and V.10, respectively; also MIL-188-114 balanced and unbalanced), and CCITT V.35. Appendix B of this document describes in detail the choices of physical interface available to the DDN subscriber and the specifications for each type of interface. Table 2.3, below, summarizes the physical interfaces available at each data rate supported by the DDN X.25 DCE, and indicates which interfaces are recommended at each signaling rate.

A DDN X.25 DTE may implement any or all of the signaling rates shown. At each signaling rate implemented, the DTE must offer at least one of the physical interface options listed as "R" (recommended) or "A" (available) for that rate in Table 2.3. Implementors are encouraged to offer the widest variety of signaling rates and physical interfaces practical to maximize the ease of use of their equipment in DDN.

Physical Interface	Signaling Rate in Kb/s									
	1.2	2.4	4.8	9.6	14.4	48	50	56	64	100
RS-232-C	R	R	R	R	R	-	-	-	-	-
RS-449 unbal. (and equiv.)	A	A	A	A	-	-	-	-	-	-
RS-449 balanced (and equiv.)	A	A	A	A	A	A	A	A	A	R
CCITT V.35	-	-	-	-	-	R	A	R	R	A

Legend

R = Recommended
A = Available
- = Not available

(Taken from Appendix B, Table B-4)

Table 2.3 DDN X.25 Physical Signaling Rates and Interfaces

3 BIBLIOGRAPHY

1. "Specification for the Interconnection of a Host and an IMP," Report No. 1822, Bolt Beranek and Newman Inc., Cambridge, MA, revision of December 1981.
2. CCITT Recommendation X.25, "Interface Between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks," International Telegraph and Telephone Consultative Committee Yellow Book, Vol. VIII.2, Geneva, 1981.
3. "Defense Data Network Subscriber Interface Guide," Defense Communications Agency, Washington, DC, July 1983.
4. "Internet Protocol Transition Workbook," SRI International, Menlo Park, CA, March 1982.
5. "Internet Protocol Implementation Guide," SRI International, Menlo Park, CA, August 1982.

APPENDIX A: DDN X.25 Implementation Details

A-1 Introduction

This Appendix serves three purposes. First, it provides information concerning the planned evolution of DDN X.25 capabilities. Second, it provides information on the use of certain DDN X.25 features and facilities at a greater level of detail than is appropriate for inclusion in the body of the DDN X.25 Interface Specification. Specifications for the use of DDN X.25 features and facilities given in this Appendix are mandatory on the part of DDN X.25 DTEs that wish to make use of these features and facilities. Finally, this Appendix presents a discussion of the limitations on the use of DDN services that will be encountered by hosts using only DDN basic X.25 service.

A-2 Operational Features of DDN X.25 DCE Releases

The capabilities of the DDN X.25 DCE will evolve over time from an initial set of capabilities to the full capabilities of this DDN X.25 Interface Specification. This section describes release-dependent features of the DDN X.25 DCE. Implementors should note that not all optional facilities of the specification will initially be available for use by DTEs.

Releases of new DCE capabilities will be compatible with DTE hardware and software implementations that meet the full DDN X.25 Interface Specification.

A-2.1 Initial Feature Support

The initial release of the DDN X.25 DCE will support flow control parameter negotiation and fast select. In addition, the DDN X.25 DCE may be configured by the DDN Administration to provide non-standard default window and packet sizes as described in CCITT X.25 Sections 7.1.2 and 7.2.1. The call precedence and type of service selection facilities will be accepted, but not acted upon, by the network. Only DDN basic X.25 service will be supported. Planned future DCE releases will support all facilities specified in FIPS 100/Federal Standard 1041 with the exception of those "additional" facilities that are listed in Section 1.2.1 of this document.

A detailed schedule of DDN X.25 DCE releases and the capabilities of each release will be supplied in a separate document.

A-2.2 Exception-Handling Procedures

Certain of the exception- or error-handling procedures of the initial release of the DDN X.25 DCE differ in detail from the procedures specified in FIPS 100/Federal Standard 1041. These differences are described below. A later release of the DDN X.25 DCE will bring these procedures into conformance. In the interim, the variances in these procedures will not preclude satisfactory operation between the DCE and a DTE, provided the DTE operates in accordance with FIPS 100/Federal Standard 1041.

A-2.2.1 Non-Octet-Aligned Data

Data packets received by the DDN X.25 DCE that are not aligned on an octet boundary are discarded at the link level. They are not passed to the DCE packet level, and no packet level diagnostic code is returned to the DTE.

A-2.2.2 RESTART REQUEST Packet

The DDN X.25 DCE will not discard, but will instead act upon, a RESTART REQUEST packet that

- (i) is too long (unless it exceeds the maximum frame size for the link level),

or

- (ii) contains a non-zero cause field.

A-2.2.3 RESET REQUEST Packet

The DDN X.25 DCE will not discard, but will instead act upon, a RESET REQUEST packet that contains a non-zero reset cause field.

A-2.2.4 CLEAR REQUEST Packet

The DDN X.25 DCE will not discard, but will instead act upon, a CLEAR REQUEST packet that contains a non-zero clearing cause field.

A-2.3 Virtual Circuit Resource Availability

In its current implementation, the DDN X.25 packet switching node is capable of supporting a minimum of one hundred simultaneous virtual circuits. As was discussed in Section 2.1.4, resources of the node are shared dynamically among the DCEs attached to the node. Therefore, no explicit guarantees are made of the number of simultaneous virtual circuits that can be made by a single DTE. Depending upon the configuration of the node, the number of simultaneous circuits supported by the node can be significantly greater than one hundred.

A-3 Detailed Features and Facilities Specifications

This section provides detailed specifications and descriptions of use for certain DDN X.25 features and facilities.

A-3.1 Additional Diagnostic Codes

The DDN X.25 DCE is capable of providing additional information to DTEs in RESTART, RESET, CLEAR INDICATION, and DIAGNOSTIC packets by means of diagnostic codes that are extensions to the set of diagnostic codes given in Annex E of CCITT Recommendation X.25. These codes are taken from the set of codes "reserved for network specific diagnostic information," and are thus not in conflict with code assignments made in Annex E. The values of these codes, and their meanings, are given in Table A-1 below.

Code Value	Meaning
128	IMP is unavailable. The packet-forwarding mechanisms of the network are unavailable to the DCE. Sent in RESET, CLEAR and RESTART packets.
130	Link level came up. Sent in RESTART and RESET packets.
131	Link level went down at remote DTE. Sent in CLEAR and RESET packets.
132	Remote DTE restarted. Sent in CLEAR and RESET packets.
133	Local resources not available for call establishment. The local DCE has too few resources to establish another call. Sent in CLEAR and DIAGNOSTIC packets.
134	Remote resources not available for call establishment. The remote DCE has too few resources to establish another call. Sent in CLEAR packets.
136	Remote host dead. The link to the remote DTE is down. Sent in CLEAR and RESET packets.
137	Remote IMP dead. The IMP to which the remote DTE is attached is down. Sent in CLEAR and RESET packets.
138	Logical subnetwork access barred. The remote DTE cannot be reached because of a communities-of-interest prohibition. Sent in CLEAR and RESET packets.
139	Connection lost. An internal error has occurred at either the remote or the local DCE which has made their virtual circuit data structures inconsistent. Sent in CLEAR and RESET packets.
140	Response lost. A response from the remote DCE failed to arrive within a reasonable time. Sent in CLEAR and RESET packets.

A-3.2 X.25 IP Interoperability Considerations

When DDN standard X.25 service is requested at call establishment (as described in Section 2.1.2.1), the call is in effect established between the DTE and a local X.25 entity. This entity subsequently extracts the IP datagrams from the X.25 data packets for transmission through the DDN Internet. This approach requires that certain conventions be followed:

1. IP datagrams are to be sent as X.25 complete packet sequences. That is, datagrams begin on packet boundaries and the M ("more data") bit is used for datagrams that are larger than one packet. Only one IP datagram is to be sent per X.25 complete packet sequence.
2. By convention, the maximum IP datagram size is 576 octets. This packet size can most efficiently be accommodated by negotiating an X.25 maximum packet size of 1024; alternatively, a DTE may use an X.25 complete packet sequence to transmit an IP datagram.
3. Because the X.25 connection is in effect terminated locally, the D and Q bits have no significance and should be set to zero.
4. The precedence bits of the IP type-of-service field are to be mapped into X.25 precedence bits (see Section 2.1.2.2) as specified in Table A-2.

IP Precedence	X.25 Precedence
000	00
001	01
010	10
011 - 111	11

Table A-2. IP Precedence to X.25 Precedence Mapping

- 141 Calling logical address not enabled or not authorized. Sent in CLEAR packets.
- 142 Calling logical name incorrect for this DTE. Sent in CLEAR packets.
- 143 Called logical name not authorized. Sent in CLEAR packets.
- 144 Called logical name not enabled. Sent in CLEAR packets.
- 145 Called logical name has no enabled DTEs. Sent in CLEAR packets.
- 146 Use of logical addresses invalid in this network. Sent in CLEAR packets.
- 147 Declared logical name now in effect. Sent in CLEAR packets.
- 148 Declared logical name was already in effect. Sent in CLEAR packets.
- 149 Declared logical name is now disabled. Sent in CLEAR packets.
- 150 Declared logical name was already disabled. Sent in CLEAR packets.
- 151 Incoming calls barred. Sent in CLEAR packets.
- 152 Outgoing calls barred. Sent in CLEAR packets.
- 192 Cleared due to higher precedence call at local DCE. Sent in CLEAR packets.
- 193 Cleared due to higher precedence call at remote DCE. Sent in CLEAR packets.
- 194 Requested precedence too high. The DTE is not authorized to establish a call at the requested precedence level. Sent in CLEAR packets.

Table A-1. Additional Packet Level Diagnostic Codes

A-3.3 The DDN Logical Addressing Facility

The DDN logical addressing facility allows references to hosts by either their physical network address or by one or more location-independent logical addresses, and allows hosts to exercise partial control over the logical address(es) by which they can be referenced. Implementation of DDN logical addressing by a host is optional.

The DDN Administration will assign seven-digit logical addresses, and will maintain a logical addressing data base. The host is then responsible for notifying the network ("enabling") of the "names" (logical addresses), if any, by which it wishes to be known. It cannot receive calls addressed to a name or originate calls under that name unless it has enabled that name. It also cannot enable a name that is not authorized for that physical address. Names can also be enabled automatically by the network, under the control of the Administration.

A-3.3.1 Logical Addresses

Logical addressing is invoked when a called address is supplied to the IMP with the flag digit F = one. The logical address consists of seven BCD digits. This name is mapped by the logical addressing facility into a DDN physical network address. The logical name need not be unique for the physical address, nor is the physical address necessarily unique for the name.

A-3.3.2 Enabling and Disabling Logical Addresses

To enable and disable logical addresses, the DDN X.25 host must send declarative CALL REQUEST packets to the DCE using a called address with the format:

ZZZZ F DDDDDDD (SS)

where the address fields are as described in Section 2.1.1. The Flag F must be set to nine, the DDN Host Identifier field specifies the logical address under consideration, and the subaddress field, which must be present, specifies the type of transaction. Declarative calls are cleared immediately by the local DCE.

If SS is zero, the logical name is enabled in normal mode; that is, that physical port will accept incoming calls to that name, and allow outgoing calls from that name. If SS is one, the logical name is disabled. If SS is two, the logical address is enabled in reverse translation mode; in this mode, the called address field of incoming call packets will be translated into a physical address (i.e., an address containing a flag F = 0), if it was given by the calling DTE (X.25 host), as a logical address (i.e., containing a flag F = 1).

Whenever a DTE comes up, or restarts, the logical names for that DTE are returned to their default state, which may be either enabled or disabled, as configured by the DDN Administration.

A-4 Limitations of DDN Basic X.25 Service

The Defense Data Network is an Internetwork environment. That is, DDN as a whole is made up of a number of constituent packet switching networks that are interconnected via gateways. Communication across gateways requires the use of the Internet Protocol, which, for a host accessing DDN using X.25, requires that the host implement the DoD standard protocol architecture and employ DDN standard X.25 service. In addition, a classified host is attached to a DDN constituent network of lower classification by means of an Internet Private Line Interface (IPLI). IPLIs, which themselves contain gateways, also require the use of the Internet Protocol; moreover, they do not, as currently designed, offer an X.25 host interface. These attributes of the DDN Internet have two implications for users of DDN basic X.25 service:

1. DDN hosts that do not implement IP and higher-level DDN protocols, and which use only DDN basic X.25 service, cannot communicate across gateways. Their network communication is therefore restricted to a single DDN constituent network.
2. X.25 hosts cannot be provided classified service on a constituent network of lower classification. Should X.25 host access be developed for the IPLI in the future, classified network access will be made available to hosts using DDN standard X.25 service only.

A-5 Derivation of DDN X.25 Addresses

All DDN hosts are assigned addresses by the Administration. The address of a DDN host may be obtained from the Network Information Center (NIC), represented as an ASCII text string in what is called "host table format." This section describes the process by which DDN X.25 addresses in the format described in Section 2.1.1 may be derived from addresses in NIC host table format.

A NIC host table address consists of the ASCII text string representations of four decimal numbers separated by periods, corresponding to the four octets of a thirty-two bit Internet address. The four decimal numbers are referred to in this section as "n", "h", "l", and "i". Thus, a host table address may be represented as "n.h.l.i". Each of these four numbers will have either one, two, or three decimal digits and will never have a value greater than 255. For example, in the host table address "10.2.0.124", n=10, h=2, l=0, and i=124. To convert a host table address to a DDN X.25 address:

1. If $h < 64$, the host table address corresponds to the DDN X.25 physical address

ZZZZ F IIIHHZZ (SS)

here:

ZZZZ = 0000
as required in Section 2.1.1.1.1;

F = 0 because the address is a physical address;

III is a three decimal digit representation of "i", right-adjusted and padded with leading zeros if required;

HH is a two decimal digit representation of "h", right-adjusted and padded with leading zeros if required;

ZZ = 00
and

(SS) is optional, as described in Section 2.1.1.1.4

In the example given above, the host table address 10.2.0.124 corresponds to the DDN X.25 physical address 000001240200.

2. If $h > 64$ or $h = 64$, the host table address corresponds to the DDN X.25 logical address

ZZZZ F RRRRRZZ (SS)

where:

ZZZZ = 0000
as required in Section 2.1.1.1.1;

F = 1 because the address is a logical address;

RRRRR is a five decimal digit representation of the result "r" of the calculation

$$r = h * 256 + i$$

(note that the decimal representation of "r" will always require five digits);

ZZ = 00
and

(SS) is optional, as described in Section 2.1.1.1.4.

Thus, the host table address 10.83.0.207 corresponds to the DDN X.25 logical address 000012145500.

In both cases, the "n" and "l" fields of the host table address are not used.

APPENDIX E: DDN Synchronous Level 1 Specification

B-1 Introduction

A host may connect to the Defense Data Network at the link level using the asynchronous bit serial protocol described in BBN Report No. 1822 as either a local host (LH) or a distant host (DH). A host may also connect to the DDN by means of a synchronous bit serial protocol at the link level, using either the method described in BBN Report No. 1822, HDH, or the DDN X.25 interface. Neither LH nor DH is recommended for new implementations.

This section describes the functional, electrical, and mechanical connection (the level 1 connection) that is required when either an HDH or an X.25 host is connected to the DDN. Hosts connecting to the DDN via HDH or X.25 require a synchronous modem connection or the equivalent, which will be supplied as part of the DDN service. The host will present the DTE interface while the DDN-provided equipment will present the DCE interface.

A long-term goal of the DDN is for all level 1 connections to be accomplished with the MIL-188-114 balanced interface. Its general equivalents are EIA RS-449/422, CCITT V.11, and Fed. Std. 1031/1020. The DDN cannot implement this at present due to the limited availability of commercial vendor hardware. In order to facilitate future DDN compatibility, all new system acquisitions should specify MIL-188-114 balanced as a required interface, in addition to an alternate interface. The selection of an alternate interface should not preclude utilization of the MIL-188-114 balanced interface when it becomes supportable.

B-2 Supported Interfaces

DDN presently supports four synchronous level 1 interfaces. They are:

1. EIA RS-232-C, CCITT V.28 & V.24;
2. MIL-188-114 balanced, EIA RS-449&422, CCITT V.11, Fed. Std. 1031/1020;
3. MIL-188-114 unbalanced, EIA RS-449&423, CCITT V.10, Fed. Std. 1031/1030; and

4. CCITT V.35.

Table B-1 is a dictionary of terms that relates the CCITT signal ID to the EIA signal ID and to the more common abbreviations. Table B-2 identifies signals as either required, optional, or not used.

Figure B-1 and Table B-3 identify typical DTE connections to the DDN. The required subscriber services will dictate which scheme is selected for a particular DTE.

Table B-4 relates required speed of service to interface type.

Together, these tables and figures serve as a guide to level 1 interface selection. From these, most systems will be able to identify the most appropriate interface. However, this information is not all-inclusive. Other interface arrangements may be possible; contact your DDN representative for assistance as required.

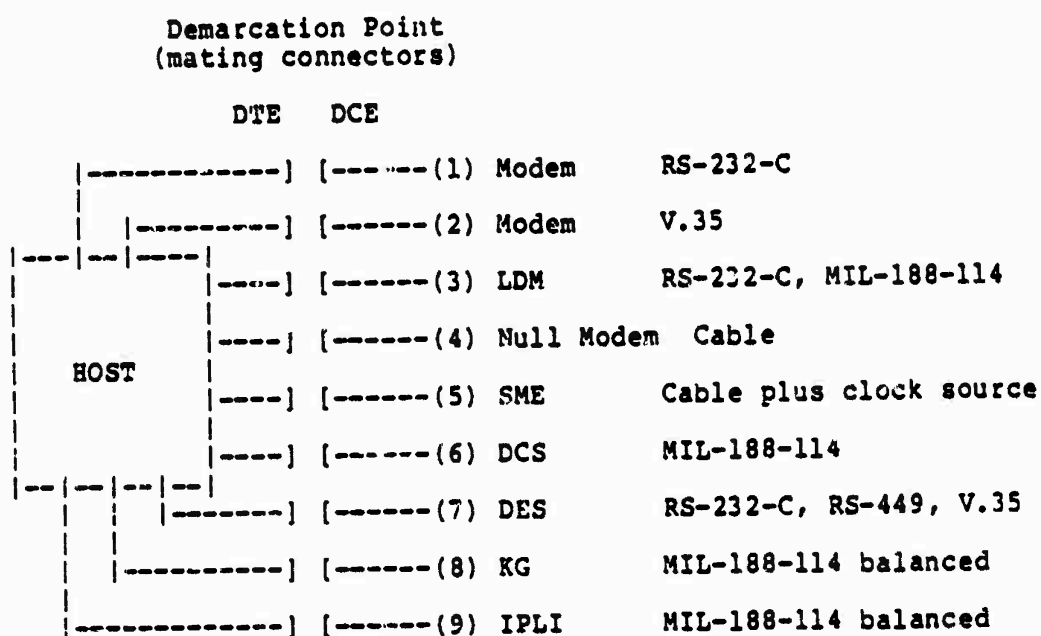


Figure B-1. Typical Level 1 Connection Schemes

EIA ID	CCITT ID	ABBREV NAME	NAME
----	-----	-----	-----
AA	101	FG	Frame (Chassis/Protective) Ground
AB	102	SG	Signal/Supply Common
SC	102a	--	RS-449 DTE Common
RC	102b	--	RS-449 DCE Common
BA	103	TD	Transmit Data
BB	104	RD	Receive Data
CA	105	RTS	Request to Send
CB	106	CTS	Clear to Send
CC	107	DSR	Data Set Ready
CD	108.2	DTR	Data Terminal Ready
CF	109	DCD	Data Carrier Detect
CG	110	SQ	Signal Quality
CH	111	--	Signal Rate Selector to DCE
CI	112	--	Signal Rate Selector to DTE
DA	113	ETC	External Transmit Clock
DB	114	TC	Transmit Clock
DD	115	RC	Receive Clock
--	116	--	Select Standby
--	117	--	Standby Indicator
SBA	118	STD	Secondary Transmit Data
SBB	119	SRD	Secondary Receive Data
SCA	120	SRS	Secondary Request to Send
SCB	121	SCS	Secondary Clear to Send
SCF	122	SCD	Secondary Carrier Detect
SCG	123	SSQ	Secondary Signal Quality
--	124	--	Select Frequency Group
CE	125	RI	Ringin Indicator
--	126	--	Select Transmit Frequency
--	127	--	Select Receive Frequency
--	128	--	External Receive Clock
--	129	RR	Request to Receive
--	130	--	Secondary Transmit Tone
--	131	--	Receive Character Timing
--	132	--	Return to Non-Data Mode
--	133	RTR	Ready to Receive
--	134	--	Received Data Present
--	136	--	New Signal
--	140	RL	Remote Loopback
--	141	LL	Local Loopback
--	142	TM	Test Status Monitor
--	191	--	Transmit Voice Answer
--	192	--	Receive Voice Answer

Table B-1. EIA and CCITT Interchange Circuits

Required: 101, 102, 103, 104, 105, 106, 107, 108.2,
109, 113, 114, and 115

Optional: 110, 125, 140, 141, and 142
(These may be required IAW future DDN
developments; it is strongly recommended
that these at least be available for
implementation upon requirement)

Not used: 111, 112, 116, 117, 118, 119, 120, 121, 122,
123, 124, 126, 127, 128, 129, 130, 131, 132,
133, 134, 136, 191, and 192

Table B-2. Signal Selection by CCITT Interchange Circuit Number

- (8) KG KG devices are U. S. Government encryption devices under strict NSA control. The requirement for security and KG devices requires the selection of the MIL-188-114 balanced interface.
- (9) Internet Private Line Interface
IPLI devices are security level community of interest isolation devices. The requirement for IPLI service requires the selection of the MIL-188-114 balanced interface.

Notes and Considerations

1. Interface (2), Modem, 48Kb/s is generally only available O-CONUS.
2. MIL-188-114 balanced is deemed equivalent to RS-449 with RS-422, the difference being that MIL-188-114 is more tolerant of noise on signal common and more tolerant of common mode noise.
3. MIL-188-114 unbalanced is deemed equivalent to RS-449 with RS-423. In most cases where MIL-188-114 balanced is specified, MIL-188-114 unbalanced is also available, but it is not recommended.
4. There are system enhancements under long term development for use in the DDN which may request additional control leads beyond those listed as required. The implementation of these enhancements will not limit operational capabilities but may impact the ability of the Network Monitoring Center to assist with host and host access line diagnosis. These enhancements may request signals from the optional category.

Table B-3. Typical Level 1 Connection Schemes

Scheme (From Fig. B-1)	Explanation
(1) Modem	RS-232 at speeds of 1200, 2400, 4800, 9600 or 14400 b/s over long haul leased voice grade telephone facilities
(2) Modem	CCITT V.35 at speeds of 48, 50, 56, 64 Kb/s over leased group (37KHz) grade facilities or in CONUS the Digital Data Service facilities.
(3) Limited Distance Modem	LDM generally available at 9600 b/s and below in an RS-232 version. Other types are available for all speeds.
(4) Null Modem	A Null Modem is a length of cable with the signal leads crossed so as to present a DCE interface. To be used in local connection schemes where either the DTE or the DCE has a clocking source capability. All four supported level 1 interfaces are available. If DTE clock and DCE clock are both available, DTE clock will be preferred.
(5) Synchronous Modem Eliminator	SME is a length of cable with a hardware device interjected. The device allows convenient crossing of signals so as to present a DCE interface. The device also provides clocking when neither the DTE nor the DCE has such capability. All four supported level 1 interfaces are available.
(6) DCS Microwave	DCS is generally a military microwave system which provides the MIL-188-114 balanced or unbalanced interfaces. It implies a speed of 50 Kbps and is usually found O-CONUS. Selection of this scheme requires selection of (4) or (5).
(7) Data Encryption Standard	DES is a commercial encryption device used by the DoD as a privacy device. DES is available with either RS-232, V.35, or RS-449/422.

Physical Interface	Signaling Rate in Kb/s									
	1.2	2.4	4.8	9.6	14.4	48	50	56	64	100
RS-232-C	R	R	R	R	R*	-	-	-	-	-
MIL-188-114 unbal. (& equiv.)	A	A	A	A	-	-	-	-	-	-
MIL-188-114 bal. (& equiv.)	A	A	A	A	A*	A	A	A	A	R**
CCITT V.35	-	-	-	-	-	R	A	R	R	A

Legend

R = Recommended
A = Available
- = Not available
* = Only available using modems
** = Only available using a local cable
connection

Table B-4. Interface Type by Service Speed

Signal Name	Abbrev	Pin No.	EIA ID	Signal Source
Frame Ground	FG	1	AA	DTE/DCE
Transmitted Data	TD	2	BA	DTE
Received Data	RD	3	BB	DCE
Request to Send	RTS	4	CA	DTE
Clear to Send	CTS	5	CB	DCE
Data Set Ready	DSR	6	CC	DCE
Signal Ground	SG	7	AB	DTE/DCE
Data Carrier Detect	DCD	8	CF	DCE
Transmit Clock	TC	15	DB	DCE
Receive Clock	RC	17	DD	DCE
Data Terminal Ready	DTR	20	CD	DTE
Ext. Transmit Clock	ETC	24	DA	DTE
Wired Spare	--	18	--	---
Wired Spare	--	22	--	---
Wired Spare	--	25	--	---

Required pins: 1, 2, 3, 4, 5, 6, 7, 8, 15, 17, 20, 24
Optional pins: 9, 10, 18, 22, 25

Notes:

1. The DTE will present a CANNON DB-25P male connector with pinouts as above or equivalent hardware with identical pinouts.
2. The DCE will present a CANNON DB-25S female connector or equivalent.

Table B-5. RS-232-C Interface

Signal Name	Abbrev	Pin Nos.	EIA ID	Signal Source
Send Data	SD	4,22	BA	DTE
Send Timing	ST	5,23	DB	DCE
Receive Data	RD	6,24	BB	DCE
Request to Send	RTS	7,25	CA	DTE
Receive Timing	RT	8,26	DD	DCE
Clear to Send	CTS	9,27	CB	DCE
Local Loopback	LL	10	--	DTE
Data Mode	DM	11,29	CC	DCE
Terminal Ready	TR	12,30	CD	DTE
Receiver Ready	RR	13,31	CF	DCE
Remote Loopback	RL	14	--	DTE
Terminal Timing	TT	17,35	DA	DTE
Test Mode	TM	18	--	DCE
Signal Ground	SG	19	AB	DTE/DCE
Receive Common	RC	20	RC	DCE
Send Common	SC	37	SC	DTE
Wired Spare	--	1	--	---
Wired Spare	--	3,21	--	---

Required pins: 4,22; 5,23; 6,24; 7,25; 8,26; 9,27;
 11,29; 12,30; 13,31; 17,35; 19; 20; 37
 Optional pins: 10; 14; 18; 1; 3,21

Notes:

1. The DTE will present a CANNON DC-37P male connector with pinouts as above or equivalent hardware with identical pinout.
2. The DCE will present a CANNON DC-37S female connector or equivalent.

Table B-6. MIL-188-114 Interface (and equivalents)

Signal Name	Abbrev	Pin Nos.	EIA ID	Signal Source
Frame Ground	FG	A	AA	DTE/DCE
Signal Ground	SG	B	AB	DTE/DCE
Transmit Data	TD	P/S	BA	DTE
Receive Data	RD	R/T	BB	DCE
Request to Send	RTS	C	CA	DTE
Clear to Send	CTS	D	CB	DCE
Data Set Ready	DSR	E	CC	DCE
Data Carrier Detect	DCD	F	CF	DCE
Local Loopback	LL	K	—	DTE
Ext. Transmit Clock	ETC	U/W	DA	DTE
Transmit Clock	TC	Y/aa	DB	DCE
Receive Clock	RC	V/X	DD	DCE

Required Pins: A; B; P/S; R/T; C; D; E; F; U/W; Y/aa;
V/X

Optional Pins: K

Notes:

1. The DTE will present a Winchester MRA(C)-34D-JTCH-H8 male connector with pinout as above or equivalent hardware with the identical pinout.
2. The DCE will present a mating female connector.

Table B-7. V.35 Interface

APPENDIX C

Federal Information
Processing Standards Publication 100

Federal Standard 1041

1983 July 6

ANNOUNCING THE JOINT STANDARD FOR
INTERFACE BETWEEN DATA TERMINAL EQUIPMENT (DTE)
AND DATA CIRCUIT-TERMINATING EQUIPMENT (DCE)
FOR OPERATION WITH PACKET-SWITCHED DATA
COMMUNICATIONS NETWORKS

Federal Information Processing Standards Publications are developed and issued by the National Bureau of Standards pursuant to section 111(f)(2) of the Federal Property and Administrative Services Act of 1949, as amended, Public Law 89-306 (79 Stat. 1127), Executive Order 11717 (38 FR 12315, dated May 11, 1973), and Part 6 of Title 15 Code of Federal Regulations (CFR).

Federal Standards in the "telecommunications" series are developed by the Office of the Manager, National Communications System. These Federal Standards are issued by the General Services Administration pursuant to the Federal Property and Administrative Services Act of 1949, as amended.

Name of Standard: Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Operation with Packet-Switched Data Communications Networks.

Category of Standard: Hardware, Data Transmission.

Explanation: Federal automated data processing equipment, services, and telecommunication equipment using public packet-switched data communications networks (PSDCN) based on the family of CCITT Recommendations derived from X.1 and X.2 shall employ the interface and protocols specified in this joint standard. In addition, designers of these internally operated and maintained Federal networks employing packet-switched technology should consider the use of this interface as appropriate. The joint standard provides:

- A family of physical layer interfaces, from which a particular interface may be selected; and
- A single data link layer control procedure; and
- Packet level procedures for virtual calls and permanent virtual circuits, and an optional datagram operation.

The mandatory interface attributes of this joint standard are summarized as follows:

PHYSICAL LEVEL

Transmission rates: 2.4, 4.8, 9.6 Kbits/s
Interface: one or more of the following: RS-232-C, X.21, RS-449

LINK LEVEL:

Procedure: LAPB
Parameter K: 7
Smallest NI: 164 Octets

FIPS PUB 100

PACKET LEVEL:

Services:	Virtual call and permanent virtual circuit
Packet types:	All basic plus Diagnostic packets. Packet Reject shall not be used.
User data field length:	Octet-aligned
Packet sequence numbering:	Modulo 8
D bit procedure:	Supported by all DCEs; DTE need not employ the D bit when sending to the DCE, but no DTE shall reject incoming packet with the D bit set to 1 or 0 as having this bit in error unless it is known by receiver that the sender has no D bit capability.
X.25 diagnostic codes:	Use standard codes whenever they apply; non-std codes may be used for events not listed in X.25 within a period of 24 months after the effective date of this standard.
Fast Select:	DCEs shall implement fast select; DTE need not employ fast select when sending to DCE, but all DTEs with higher level functionality which allows response to fast select must be able to accept incoming fast select packet.
Interrupt packet:	Receipt of a DTE interrupt packet before a previous DTE interrupt packet has been confirmed is an error condition.
Duplicated facility codes:	The last appearing facility code should be treated by the DTE as if it were the only appearance of that code.
Non-zero cause field of restart request packet:	Discarded
Restart request too long in state rl:	Discarded

This joint standard is intended to enhance interoperability by specifying certain subsets and other constraints on Federal use of CCITT Recommendation X.25.

The Government's intent in employing this joint standard is to reduce the cost of acquiring and using Federal automated data processing equipment, services, and telecommunication equipment with PSDCN. The joint standard is also intended to reduce the cost of acquiring and using Government-owned or leased PSDCN. These goals will be achieved by:

- Increasing the available alternative sources of supply;
- Increasing the reutilization of Government resources; and,
- Assuring the required interoperability.

Approving Authority: Secretary of Commerce (Federal Information Processing Standards). Administrator, General Services Administration (Federal Standards).

Maintenance Agency: The National Bureau of Standards and the Office of the Manager, National Communications System will jointly maintain this standard coordinating as necessary with the General Services Administration (GSA).

Cross Index: The following are related standards upon which this FIPS PUB is based. The inclusion of a particular standard on this list does not necessarily mean that the standard is applicable in all cases to which this FIPS PUB applies.

(a) International Standard 2110-1980: *Data Communication—25-Pin DTE/DCE Interface Connector and Pin Assignments*.

(b) International Telegraph and Telephone Consultative Committee (CCITT) Recommendation V.24 (1980): *List of Definitions for Interchange Circuits Between Data Terminal Equipment and Data Circuit Terminating Equipment*.

(c) CCITT Recommendation V.28 (1980): *Electrical Characteristics for Unbalanced Double-Current Interchange Circuits*.

(d) Electronics Industries Association (EIA) RS-232-C (1969 August): *Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange*.

(e) International Standard 4902-1980: *Data Communication—37-Pin and 9-Pin DTE/DCE Interface Connectors and Pin Assignments*.

(f) CCITT Recommendation V.11 (X.27) (1980): *Electrical Characteristics for Balanced Double-Current Interchange Circuits for General Use with Integrated Circuit Equipment in the Field of Data Communications*.

(g) EIA RS-422-A (1978 June): *Electrical Characteristics of Balanced Voltage Digital Interface Circuits*.

(h) Federal Standard 1020A (1980 January): *Telecommunications: Electrical Characteristics of Balanced Voltage Digital Interface Circuits*.

(i) CCITT Recommendation V.10 (X.26) (1980): *Electrical Characteristics for Unbalanced Double-Current Interchange Circuits for General Use with Integrated Circuit Equipment in the Field of Data Communications*.

(j) EIA RS-423-A (1978 June): *Electrical Characteristics of Unbalanced Voltage Digital Interface Circuits*.

(k) Federal Standard 1030A (1980 January): *Telecommunications: Electrical Characteristics of Unbalanced Voltage Digital Interface Circuits*.

(l) CCITT Recommendation X.21bis (1980): *Use on Public Data Networks of Data Terminal Equipment which are Designed for Interfacing to Synchronous V-Series Modems*.

(m) CCITT Recommendation V.34 (1980): *Loop Test Devices for Modems*.

(n) EIA RS-449 (1977 November): *General Purpose 37-Position and 9-Position Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment*.

(o) Federal Standard 1031 (1980 June): *Telecommunications General Purpose 37-Position and 9-Position Interface Between Data Terminal Equipment and Data Circuit Terminating Equipment*. (Implementing instructions in the form of a Federal Property Management Regulation have not yet been issued. The General Services Administration is considering cancelling FED-STD 1031. Furthermore, a Federal Information Processing Standard for ADP applications corresponding to Federal Standard 1031 has not been adopted by the National Bureau of Standards.)

(p) International Standard 4903-1980: *Data Communication—15-Pin DTE/DCE Interface Connector and Pin Assignments*.

(q) EIA Industrial Electronics Bulletin No. 12 (1977 November): *Application Notes on Interconnection Between Interface Circuits Using RS-449 and RS-232-C*.

(r) Draft International Standard 2593 (1980): *Data Communication—34-Pin DTE/DCE Interface Connector and Pin Assignments*.

(s) CCITT Recommendation V.35 (1980): *Data Transmission at 48 Kilobits per Second Using 60-108 kHz Group Band Circuits*.

(t) CCITT Recommendation X.21 (1980): *General Purpose Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment for Synchronous Operation on Public Data Networks*.

(u) CCITT Recommendation V.5 (1980): *Standardization of Data-Signalling Rates for Synchronous Data Transmission in the General Switched Telephone Network*.

(v) CCITT Recommendation V.6 (1980): *Standardization of Data-Signalling Rates for Synchronous Data Transmission on Leased Telephone-Type Circuits*.

(w) American National Standard X3.1-1974: *Synchronous Signaling Rates for Data Transmission*.

(x) Federal Information Processing Standard Publication 22-1 (1977 September): *Synchronous Signalling Rates Between Data Terminal and Data Communication Equipment*. (FIPS PUB 22-1 is identified also as FED-STD 1013.)

FIPS PUB 100

(y) Federal Standard 1013 (1977 August): *Telecommunications: Synchronous Signaling Rates Between Data Terminal Equipment and Data Circuit-Terminating Equipment Utilizing 4 kHz Circuits* (FED-STD 1013 is identified also as FIPS PUB 22-1.)

(z) American National Standard X3.36-1975: *Synchronous High-Speed Data Signaling Rates Between Data Terminal Equipment and Data Communication Equipment*.

(aa) Federal Information Processing Standards Publication 37 (1975 June): *Synchronous High Speed Data Signaling Rates Between Data Terminal Equipment and Data Communications Equipment* (FIPS PUB 37 is identified also as FED-STD 1001.)

(ab) Federal Standard 1001 (1975 June): *Telecommunications: Synchronous High-Speed Data Signaling Rates Between Data Terminal Equipment and Data Communications Equipment* (FED-STD 1001 is identified also as FIPS PUB 37.)

(ac) EIA RS-269-B (1976 January): *Synchronous Signaling Rates for Data Transmission*.

(ad) International Standard 3309-1979: *Data Communication—High-Level Data Link Control Procedures—Frame Structure*.

(ae) International Standard 4335-1979: *Data Communication—High-Level Data Link Control Procedures—Elements of Procedures*.

(af) Addendum 1 to International Standard 4335-1979: *Data Communication—High-Level Data Link Control Procedures—Elements of Procedures*.

(ag) Addendum 2 to International Standard 4335-1979: *Data Communication—High-Level Data Link Control Procedures—Elements of Procedures*.

(ah) International Standard 6256-1980: *Data Communication—High-Level Data Link Control Procedures—Balanced Class of Procedures*.

(ai) American National Standard X3.66-1979: *Advanced Data Communication Control Procedures (ADCCP)*.

(aj) Federal Information Processing Standards Publication 71 (1980 May) as revised by the Federal Register notice 47 FR 23798, dated June 1, 1982 and corrected by the notice 47 FR 25397 dated June 11, 1982: *Advanced Data Communication Control Procedures (ADCCP)* (FIPS PUB 71 is technically consistent with FED-STD 1003A.)

(ak) Federal Information Processing Standards Publication 78 (1980 September): *Guideline for Implementing Advanced Data Communication Control Procedures (ADCCP)*.

(al) Federal Standard 1003A (1981 August): *Telecommunication: Synchronous Bit-Oriented Data Link Control Procedures* (FED-STD 1003A is technically consistent with FIPS PUB 71.)

(am) CCITT Recommendation X.25 (1980): *Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks*.

(an) Draft Proposed International Standard 7498: *Data Processing—Open Systems: Interconnection—Basic Reference Model*.

(ao) CCITT Recommendation X.1 (1980): *International User Classes of Service in Public Data Networks*.

(ap) CCITT Recommendation X.2 (1980): *International User Facilities in Public Data Networks*.

(aq) CCITT Recommendation X.96 (1980): *Call Progress Signals in Public Data Networks*.

Applicability: The technical specifications of this joint standard shall be employed in the acquisition, design, and development of all Federal automated data processing equipment, services, and telecommunication equipment and PSDCN whenever an interface based on CCITT Recommendation X.25 (1980), *Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Networks*, is required. Referred to below as CCITT Recommendation X.25, Recommendation X.25, or X.25.

Implementation: The provisions of this joint standard are effective July 4, 1983. Any applicable equipment or service ordered on or after the effective date, or procurement action for which solicitation documents have not been issued by that date, must conform to the provisions of this standard unless a waiver has been granted in accordance with the procedures described below.

This joint standard shall be reviewed by the Institute for Computer Sciences and Technology, National Bureau of Standards and the Office of the Manager, National Communications System, within five years after its effective date. This review shall take into account technological trends and other factors to determine if the joint standard should be affirmed, revised, or withdrawn.

Specifications: This joint standard adopts a subset, identified below, of the International Telegraph and Telephone Consultative Committee's Recommendation X.25.

(a) At the physical level, the provisions of Section 1 of CCITT Recommendation X.25 shall be used. As a minimum, networks shall support dedicated circuit access; other types of access (e.g., through the general switched telephone network) may also be offered.

CCITT Recommendation X.1 standardizes data signalling rates of 2.4, 4.8, 9.6, and 48 kbits/s for packet mode interfaces. At a minimum, networks shall support the synchronous data signalling rates of 2.4, 4.8, and 9.6 kbits/s full duplex; other speeds (e.g., 19.2 kbits/s) may also be offered. The 48 kbits/s rate need not be supported in those locations where it is not available; 56 kbits/s is recommended in its place (see American National Standard X3.36-1975 and related documents referenced above). The term "user class of service" used in X.25 refers to the data signalling rate of DTE/DCE interface.

In accordance with CCITT Recommendation X.25, networks shall provide one or more of the following interface options:

- i. CCITT Recommendation X.21;
- ii. EIA RS-232-C, which is essentially equivalent to one of the options in CCITT Recommendation X.21bis;
- iii. CCITT Recommendation X.21bis option that is equivalent to RS-449 using only the EIA RS-423A unbalanced electrical characteristics.

Interworking between EIA RS-232-C on one side of the interface and RS-449 on the other side is permitted in accordance with EIA Industrial Electronics Bulletin Number 12. Where interworking with RS-232-C equipment is not required, the provisions described below employing RS-449 with the RS-422A electrical characteristics may optionally be employed at signalling rates below 48 kbit/s.

Networks which support 48 or 56 kbits/s data signalling rates shall provide one or more of the following interface options:

- i. CCITT Recommendation X.21;
- ii. CCITT Recommendation X.21bis option that specifies CCITT Recommendation V.35; or
- iii. CCITT Recommendation X.21bis option that specifies CCITT Recommendation V.36 which is equivalent to EIA RS-449.

NOTE: Current study in national and international standards groups may result in the development of additional physical interfaces. Each such physical interface will be evaluated for inclusion in this joint standard. If there are significant savings, one physical interface may be selected as the future mandatory physical interface.

NOTE: DTE purchasers and designers should determine which physical interface(s) is provided by the associated DCE(s).

(b) Only the LAPB link level procedures shall be used.

NOTE: These procedures are a subset of those described in FIPS PUB 71 and Federal Standard 1003A and correspond to FIPS PUB 78 recommended class B. This subset is identified as follows:

- i. Link configuration: two combined stations on a point-to-point link.
- ii. Class of procedures: balanced asynchronous (BA) with options two and eight. The RSET command shall not be used. (RSET is found in option 11 of the FIPS PUB 71. RSET is part of the basic repertoire in Federal Standard 1003A; option 11 of Federal Standard 1003A deletes the RSET command. Note that RSET is *not* part of CCITT Recommendation X.25.)
- iii. Two-way simultaneous operation shall be employed.
- iv. The smallest NI, (the maximum number of bits in an information frame excluding flags and zero bit insertion for transparency), which shall be supported shall be 164 octets (the maximum length of

FIPS PUB 100

fast select call setup packet). If a DTE neither transmits, nor receives for processing by higher level functionality fast select packets, an N1 as small as 135 octets may be supported by the DTE.

v. The address of the combined station provided by the network shall be 10000000; the address of the other combined station shall be 11000000; where the left-hand bit is the least significant bit (bit number 1) and shall be transmitted first. This convention is consistent with the provisions of FIPS 71 and Federal Standard 1003A.

vi. The FCS shall be a 16-bit sequence as indicated in Section 2.2.7. DTE/DCE may also employ the 32-bit FCS as indicated in FIPS PUB 71 (revised) and FED-STD 1003A. DTE/DCE equipment using the 32-bit FCS shall be able to also operate with the 16-bit FCS. The smallest N1 shall be 166 octets when the 32-bit FCS is used. If a DTE neither transmits, nor receives for processing by higher level functionality fast select packets, an N1 as small as 137 octets may be supported by the DTE when the 32-bit FCS is used.

NOTE: FIPS PUB 78 provides a detailed discussion of the relative merits of the 16-bit and 32-bit FCS.

vii. The frame reject information field shall be padded with 4 zero bits in bit positions 21 through 24 of the information field to provide a length of three octets.

viii. It is required that all implementations be capable of operating with $K=7$; optionally, values of 1 to 6 are permissible with modulo 8 operation and values 1 to 127 are permissible with modulo 128 operation.

NOTE: DTE purchasers and designers should determine that values of k other than 7 are supported by the associated DCE(s).

(c) The user data field of packets shall be an integral number of octets. If a packet is received which shows a user data field not equal to an integral number of octets, the receiving DTE/DCE shall follow the packet level procedures for processing a packet type which is too long. A new diagnostic code, "non-octet aligned packet," consistent with the *Data Communications—X.25 Packet Layer Specification for Data Terminal Equipment*, ISO DP 8208, November 8, 1982, is recommended as #82.

(d) The reject packet shall not be used.

(e) All DCE restart confirmation, DCE reset confirmation, and DCE clear confirmation packets shall be interpreted by the DTE as having local significance only.

(f) The D-bit shall be implemented by all networks. DTE's need not employ the D-bit procedures when transmitting to the network, but no DTE shall reject incoming packets with the D-bit set to 1 or 0 as having this bit in error unless the receiving DTE knows the remote DTE has not implemented the D-bit procedure; in this case, the receipt of a D-bit set to 1 may be treated by the receiving DTE as an error condition.

(g) The selection of logical channel number for new virtual calls shall follow the procedures suggested in Section 4.1.2 Note 2, Annex A Note 5, and Annex A Note 6, of the CCITT Recommendation X.25.

(h) It is required that all implementations be capable of operating with packet sequence numbering modulo 8; optionally, implementations of packet sequence numbering modulo 128 are also permitted.

NOTE: DTE purchasers and designers should determine if the associated DCE(s) support packet sequence numbering modulo 128.

(i) All DTE's and DCE's shall follow the flow control principles outlined in the first two sentences of the first paragraph of Section 4.4.1.3 of CCITT Recommendation X.25.

(j) The alternative procedure for passing packets containing a P(S) that is out of sequence but within the window as described in the third paragraph of Section 4.4.1.3 of CCITT Recommendation X.25 shall not be used.

(k) The second sentence of Section 4.4.1.4 Note 2 shall not apply. This sentence permits networks to defer updating the window for data packets with $D=0$, and sent within the window but before a data packet with $D=1$, until the network receives a corresponding P(R) for the packet with $D=1$.

(l) The resetting cause field of a reset request packet shall be set to zero. If a reset request is received with a non-zero resetting cause field, the packet shall be discarded. The network shall then initiate the resetting procedure with the resetting cause field indicating local/remote procedure error.

(m) The clearing cause field of a clear request packet shall be set to zero. If a clear request packet is received with a non-zero clearing cause field, the packet shall be discarded. The network shall then initiate the clearing procedure with the clearing cause field indicating local/remote procedure error.

(n) The restarting cause field of a restart request packet shall be set to zero. If a restart request packet is received with a non-zero restart cause field, the restart request packet shall be discarded without further action. Optionally, the DCE may generate a diagnostic packet with a recommended diagnostic code #81 (improper cause code from DTE), which is consistent with the *Data Communication—X.25 Packet Layer Specification for Data Terminal Equipment*, ISO DP 8208, November 8, 1982.

(o) A diagnostic code shall be provided in all clear request, reset request, and restart request packets in accordance with the codes listed in Annex E of CCITT Recommendation X.25 whenever they apply; non-assigned codings in X.25 may be used for events not listed in X.25 within the period of 24 months after the effective date of this standard. Prior to the end of this 24 month period, this standard will be reviewed by NBS to determine whether the standard should be revised to incorporate a different table. After this revision, codes not specifically listed shall not be used.

(p) A generic diagnostic code shall not be used when a more specific diagnostic code is known to be applicable.

(q) The network diagnostic codes shall be used in accordance with the codes listed in Annex E of CCITT Recommendation X.25 whenever they apply; non-assigned codings in X.25 may be used for events not listed in X.25 within the period of 24 months after the effective date of this standard. Prior to the end of this 24 month period, this standard will be reviewed by NBS to determine whether the standard should be revised to incorporate a different table. After this revision, network diagnostic codes not specifically listed shall not be used.

(r) The network shall consider the receipt of a DTE interrupt packet before a previous DTE interrupt packet has been confirmed as an error, and shall execute the error procedure described in Annex C, Table C-4/X.25 and the corresponding note 2.

(s) The timeouts and time limits specified in Annex D shall be observed by all DTE and DCE equipment. T21 shall not be less than the value given in table D-2/X.25. The preferred actions listed in table D-2/X.25 shall be followed.

(t) When the link level procedures enter the logically disconnected state, the associated packet level procedures shall clear all virtual calls and reset all permanent virtual circuits and datagram logical channels. When the link level procedures reenter the information transfer state, the associated packet level procedures shall execute the restart procedure. The terms "logically disconnected state" and "information transfer state" are used as defined in American National Standard X3.66-1979 (referenced above). Link level procedures enter the logically disconnected state when a DISC command is sent and a UA response is received, for example. The link level procedure shall also be considered to be in the logically disconnected state after N2 (re)transmissions of SABM or DISC, where N2 is as defined in CCITT Recommendation X.25. The logically disconnected state is not assumed after N2 (re)transmissions of other types of frames.

(u) If a restart request packet is received in state r1 which exceeds the maximum permitted length, the DCE shall discard the restart request packet without further action. Optionally, the DCE may generate a diagnostic packet with diagnostic code #39 (packet too long).

(v) In the event that a facility code appears more than once in a facility field, the receiving DTE detecting this condition should treat the last appearance of the particular code as if it were the only appearance of that code.

(w) All networks shall supply diagnostic packets when their use is suggested in CCITT Recommendation X.25. No DTE shall reject diagnostic packets as errors.

(x) In Section 6.1.1, the second paragraph, the last phrase, "and is set to 0 in all other packets", shall be interpreted that the Qualifier bit is set to 0 in all other packets except data packets. For the case of data packets, the Qualifier bit is set to 0 or 1 as indicated in Section 4.3.6 of CCITT Recommendation X.25.

FIPS PUB 100

(y) The list of user facilities for packet-switched data networks, extracted from CCITT Recommendation X.2, is given below. These facilities are described in Section 7 of CCITT Recommendation X.25. The following further constraints apply:

- i. Networks shall provide the facilities designated as essential "E" below.
- ii. Networks shall also implement the Fast Select and Fast Select Acceptance facilities to facilitate more efficient operation in conveying higher layer protocol information or user data during call establishment. DTE's need not employ fast select packets when transmitting to the network, but all DTE's associated with the higher level functionality which allows response to a fast select packet must be able to accept incoming fast select packets.
- iii. The packet retransmission facility shall not be used.
- iv. All DTE's which employ any of the facilities labelled as additional "A" below (except Fast Select and Fast Select Acceptance) shall also be capable of operating without employing any A facilities (except Fast Select and Fast Select Acceptance).
- v. The throughput class value of 48,000 bits/s may be interpreted as 56,000 bits/s in those locations where 56,000 bits/s access is used.

Facilities of packet-switched data networks:

<i>User Facility</i>	<i>VC</i>	<i>PVC</i>	<i>DG*</i>
Optional user facilities assigned for an agreed contractual period:			
Extended packet sequence numbering (module)	A	A	A*
Non-standard default window sizes	A	A	A*
Non-standard default packet sizes 16, 32, 64, 256, 512, 1024	A	A	-
Default throughput class assignment	A	A	A*
Flow control parameter negotiation	E	-	-
Throughput class negotiation	E	-	-
Packet retransmission	A***	A***	A***
Incoming calls barred	E	-	E*
Outgoing calls barred	E	-	E*
One-way logical channel outgoing	E	-	A*
One-way logical channel incoming	A	-	A*
Closed user group	E	-	E*
Closed user group with outgoing access	A	-	A*
Closed user group with incoming access	A	-	A*
Incoming calls barred within a closed user group	A	-	A*
Outgoing calls barred within a closed user group	A	-	A*
Bilateral closed user group	A	-	A*
Bilateral closed user group with outgoing access	A	-	A*
Reverse charging acceptance	A	-	A*
Fast select acceptance	A**	-	-
Datagram queue length selection*	-	-	A*
Datagram service signal logical channel*	-	-	A*
Datagram non-delivery indication*	-	-	E*
Datagram delivery confirmation*	-	-	E*
D-bit modification	A	A	-
<i>Optional user facilities requested by the DTE on a per call basis</i>			
Closed user group selection	E	-	E*
Bilateral closed user group selection	A	-	A*
Reverse charging	A	-	A*
RPOA selection	A	-	A*
Flow control parameter negotiation	E	-	-
Fast select	A**	-	-
Throughput class negotiation	E	-	-
Abbreviated address calling	FS	-	A*
Datagram non-delivery indication	-	-	E*
Datagram delivery confirmation	-	-	E*

NOTE: Detailed explanations of these facilities are provided in CCITT Recommendation X.25.

FIPS PUB 100

LEGEND:

- E = An essential user facility to be offered by all networks.
- A = An additional user facility which may be offered by certain networks.
- FS = Further study is required. This standard will be modified when this study is complete.
- = Not applicable.
- DG = Applicable when the datagram service is being used.*
- VC = Applicable when the virtual call service is being used.
- PVC = Applicable when the permanent virtual circuit service is being used.
- * - The datagram service and its related facilities *may* be used *only* when:
 - there is to be a one-way transfer of information which does not require recovery at the network layer; and,
 - a response to this transfer of information is not required at the network layer.

NOTES: 1. At the present time, the transfer of datagram packets across international borders through public packet-switching networks is not permitted.

2. DCE's are not required to provide datagram service. DTE's are not required to generate or accept datagrams and datagram-related packets.

** - Fast select shall be provided by all DCE's. All DTE's associated with the higher level functionality which allows response to a fast select packet must be capable of accepting incoming fast select packets, but need not generate fast select packets.

*** - The packet retransmission facilities shall not be used.

(z) The list of the applicable call progress signals, extracted from CCITT Recommendation X.96, is given below. These signal definitions apply to the cause codes specified in CCITT Recommendation X.25. The related circumstances giving rise to each call progress signal is also defined in table 1 below. The significance of categories indicates broadly the type of action expected of the DTE receiving the signal:

<i>Category</i>	<i>Significance</i>
A	Requested action confirmed by network.
B	Call cleared because the procedure is complete.
C1 and C2	Call cleared. The calling DTE should call again soon: the next attempt may be successful. However, after a number of unsuccessful call attempts with the same response, the cause could be assumed to be in Category D1 or D2. The interval between successive attempts and the number of maximum attempts will depend on a number of circumstances including: <ul style="list-style-type: none"> - nature of the call progress signal - users' traffic pattern - tariffs - possible regulations by the network provider. OR Reset. The DTE may continue to transmit data recognizing that data loss may have occurred.
D1 and D2	Call cleared. The calling DTE should take other action to clarify when the call attempt might be successful. OR Reset (for permanent virtual circuit only). The DTE should cease data transmission and take other action as appropriate.
C1 and D1	Due to subscriber condition.
C2 and D2	Due to network condition.

The sequence of call progress signals in table 1 implies, for Categories C and D, the order of call set-up processing by the network. In general, the DTE can assume, on receiving a call progress signal, that no condition higher up in the table is present. Network congestion is an exception to this general rule. The actual coding of call progress signals does not necessarily reflect this sequence.

Users and DTE manufacturers are warned to make due allowance to possible later extensions to this table by providing appropriate fall-back routines for unexpected signals.

FIPS PUB 100

Table 1

<i>Call Progress Signal</i>	<i>Definition</i>	<i>Category</i>
Delivery confirmation	The datagram has been accepted by the destination DTE.	A
Local procedure error	A procedure error caused by the DTE is detected by the DCE at the local DTE/DCE interface.	C1
Network congestion	A condition exists in the network such as: 1) temporary network congestion 2) temporary fault condition within the network, including procedure error within a network or an international link.	C2
Invalid facility request	A facility requested by the calling DTE is detected as invalid by the DCE at the local DTE/DCE interface. Possible reasons include: - request for a facility which has not been subscribed to by the DTE; - request for a facility which is not available in the local network; - request for a facility which has not been recognized as valid by the local DCE.	D1 or D2
RFOA out of order	The RFOA nominated by the calling DTE is unable to forward the call.	D2
Not obtainable	The called DTE address is out of the numbering plan or not assigned to any DTE.	D1
Access barred	The calling DTE is not permitted the connection to the called DTE. Possible reasons include: - unauthorized access between the calling DTE and the called DTE; - incompatible closed user group.	D1
Reverse charging acceptance not subscribed	The called DTE has not subscribed to the reverse charging acceptance facility.	D1
Fast select acceptance not subscribed	The called DTE has not subscribed to the fast select acceptance facility.	D1
Incompatible destination	The remote DTE/DCE interface or the transit network does not support a function or facility requested (e.g., the datagram service).	D1
Out of order	The remote number is out of order. Possible reasons include: - DTE in Uncontrolled Non Ready; - DCE Power Off; - Network fault in the local loop; - X.25 Level 1 not functioning; - X.25 Level 2 not in operation.	D1 or D2

Table 1 (Continued)

<i>Call Progress Signal</i>	<i>Definition</i>	<i>Category</i>
Number busy	The called DTE is detected by the DCE as engaged on other call(s), and therefore as not being able to accept the incoming call. (In the case of the datagram service, the queue at the destination DCE is full.)	C1
Remote procedure error	A procedure error caused by the remote DTE is detected by the DCE at the remote DTE/DCE interface.	D1
Network operational	Network is ready to resume normal operation after a temporary failure or congestion.	C1
Remote DTE operational	Remote DTE/DCE interface is ready to resume normal operation after a temporary failure or out of order condition (e.g., restart at the remote DTE/DCE interface. Loss of data may have occurred.	C1 or D1
DTE originated	The remote DTE has initiated a clear, reset, or restart procedure.	B or D1

Waivers: Waiver of this standard is required when an interface based on CCITT Recommendation X.25 (1980) is to be employed and has either one of the following conditions: 1) The interface has options that are not permitted by this standard; 2) The interface does not implement all options mandated by this standard.

Heads of agencies desiring a waiver from the requirements stated in this standard, so as to acquire applicable equipment or service not conforming to this standard, shall submit a request for waiver to the Administrator, General Services Administration, for review and approval. Approval will be granted if, in the judgment of the Administrator after consultation with the Assistant Secretary of Commerce for Productivity, Technology and Innovation, based on all available information including that provided in the waiver requests, a major adverse economic or operational impact would occur through conformance with this standard.

A request for waiver shall include a justification for the waiver, including a description and discussion of the adverse economic or operational impact that would result from conforming to this standard as compared to the alternative for which the waiver is requested. ICST and NCS will provide technical assistance, as required, to GSA.

Where to Obtain Copies: Copies of this publication are for sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. When ordering, refer to Federal Information Processing Standards Publication 100 (FIPS-PUB-100)/Federal Standard 1041 (FED-STD 1041), and title. When microfiche is desired, this should be specified. Payment may be made by check, money order, purchase order, credit card, or deposit account.

The CCITT X.25 specifications upon which this publication is based may also be obtained from NTIS. Specify PB82-187766; the cost is \$50; telephone (703) 487-4650.

Network Working Group
Request for Comments: 929

Joel Lilienkamp (SDC)
Richard Mandell (SDC)
Michael Padlipsky (Mitre Corp.)
December 1984

PROPOSED HOST-FRONT END PROTOCOL

Status Of This Memo

The reader should be aware of several things in regard to what the present document is up to. First and foremost, IT IS A PROPOSAL FOR A STANDARD, NOT A STANDARD ITSELF. Next, it assumes that the separate document, RFC 928, which is an introduction to the present document, has been read before it is. Next, it should be understood that "final cut" over this version of the document has been exercised by the author of RFC 928, not by the primary author of the present document, so any readers bothered by style considerations should feel free to blame the former, who's used to it, rather than the latter, who may well be guiltless. (Editing at a distance finally become too hard to manage, so if I'm typing it myself I'm going to fiddle with it myself too, including, but not limited to, sticking my own section on the Conceptual Model in before Joel's words start, rather than leaving it in the Introduction. MAP)

Finally, it should be noted that this is not a finished document. That is, the intent is eventually to supply appendices for all of the protocol offloadings, describing their uses of protocol idiosyncratic parameters and even their interpretations of the standard per-command parameters, but in order to get what we've got into circulation we haven't waited until all such appendices have been written up. (We do have notes on how to handle FTP, e.g., and UDP will be pretty straightforward, but getting them ready would have delayed things into still another calendar year, which would have been very annoying ... not to say embarrassing.) For that matter, it's not even a finished document with respect to what is here. Not only is it our stated intention to revise the protocol based upon implementation experience gained from volunteer test implementations, but it's also the case that it hasn't proven feasible to iron out all known wrinkles in what is being presented. For example, the response codes almost certainly need clarification and expansion, and at least one of us doesn't think mandatory initial parameters need control flags. However, to try too hard for polish would be to stay in subcommittee for the better part of forever, so what you see is what we've got, but certainly isn't meant to be what you or we are stuck with.

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Lilienkamp & Mandell & Padlipsky

[Page 1]

RFC 929
Proposed Host-Front End Protocol

December 1984

Conceptual Model

There are two fundamental motivations for doing outboard processing. One is to conserve the Hosts' resources (CPU cycles and memory) in a resource sharing intercomputer network, by offloading as much of the required networking software from the Hosts to Outboard Processing Environments (or "Network Front-Ends") as possible. The other is to facilitate procurement of implementations of the various intercomputer networking protocols for the several types of Host in play in a typical heterogeneous intercomputer network, by employing common implementations in the OPE. A third motivation, of basing a network security approach on trusted mandatory OPEs, will not be dealt with here, but is at least worthy of mention.

Neither motivation should be allowed to detract from the underlying, assumed desire to perform true intercomputer networking, however. Therefore, it is further assumed that OPEs will be attached to Hosts via a flexible attachment strategy, as described in [1]. That is, at the software level an explicit Host-Front End Protocol (H-FP) will be employed between Hosts and OPEs, rather than having OPEs emulate devices or device controllers already "known" to Host operating systems (in order to avoid introducing new code into the Host).

For reasons discussed in the Introduction, an H-FP resolves into three layers. The Link layer enables the exchange of bits between Host and OPE. The Channel layer enables the bit streams to be demultiplexed and flow controlled (both the Channel and Link layers may use preexisting per-Host mechanizations, it should be recalled). The Command (or "Service Access") layer is our primary concern at present. It serves as the distributed processing mechanism which allows processes on Hosts to manipulate protocol interpreters (PIs) in OPEs on their behalf; for convenience, it will be referred to as "the H-FP" here. (It should be noted that the Link and Channel layers may be viewed as roughly equivalent to the inboard processing investment for a Host-comm subnet processor PI and device driver, so in practical terms the savings of resources achieved by outboard processing come from making the H-FP "smaller" than the inboard implementations of the protocols it allows to be offloaded.)

The crucial property of the H-FP conceptually is that it stands as the interface between a (Host) process and a PI (which is actually outboard). Usually, the model is that of a closed subroutine interface, although in some cases an interprocess communication mechanism model must be appealed to. That is, the interactions between cooperating H-FP PIs in some sense mimic subroutine or IPC calls, from the perspective of Host processes calling upon their own H-FP PIs, which in turn are of course interfacing via just such

RFC 929
Proposed Host-Front End Protocol

December 1984

mechanisms themselves. Another way of putting it is that "if the protocols were inboard," the processes invoking H-FP wouldn't know the difference. H-FP, then, may be viewed as a roundabout way of letting Host processes "get at" various PIs.

Naturally, the mechanization of the desired concept cannot be particularly literal. After all, the Hosts and the OPEs are different processors, so we're not envisioning a passing through of parameters in an exact fashion. However, in broad terms the model is just that of a somewhat funny interface between a process and a PI. (This should not be construed as ruling out the occurrence of events which prompt the OPE to initiate an exchange of commands with the Host, though; see the Introduction for more on the topic of "Symmetric Begins.")

Interaction Discipline

The interaction between the Host and the OPE must be capable of providing a suitable interface between processes (or protocol interpreters) in the Host and the off-loaded protocol interpreters in the OPE. This interaction must not, however, burden the Host more heavily than would have resulted from supporting the protocols inboard, lest the advantage of using an OPE be overridden.

Channel Level Interaction

As stated elsewhere, the Channel level protocol (implicitly in conjunction with the Link level) provides two major functions. These are demultiplexing the traffic from the Link level into distinct data streams, and providing flow control between the Host and the OPE on a per stream basis. These hold even if the Host-OPE attachment is DMA.

The data streams between the Host and the OPE are bidirectional. In this document, the basic unit of data transferred by the Channel level is referred to as a "chunk". The primary motivation for this terminology is that the H-FP permits the Channel level to be one of several possible protocols, each with its own terminology. For example, a chunk on an X.25 Channel would be a packet, while a chunk on the DFI H-FP channel would be a message. While the Command level is, in a sense, "more efficient" when the chunk size is permitted to be large, the flexibility permitted in the choice of protocols at the Channel level precludes any assumptions about the chunk size.

Each data stream is fully asynchronous. A Channel protocol user can send data at any time, once the channel has been properly opened. (The Command level's logic may render some actions meaningless, however.) The data transfer service provided by the Channel protocol

Lillienkamp & Mandell & Padlipsky

[Page 3]

RFC 929

December 1984

Proposed Host-Front End Protocol

is reliable; this entails delivery in the correct order, without duplication, and checked for bit errors. All retransmission, error checking, and duplicate detection is provided by this protocol in a way that is transparent to the user. (If the attachment is DMA, stream identification and chunk length must still be provided for.)

The flow control at the Channel level is provided to prevent the OPE and the Host from overloading each other's resources by excessive transmissions. In general, this flow control should not directly affect the outboard protocol interpreters' operation. On the other hand, this flow control has the same effect as explicit interface events that provide flow control between the user and the protocol interpreter (e.g., the Allocate event of the interface specification for TCP found in MIL-STD 1778). Hence, such events do not need to be communicated explicitly at the Command level. (If the attachment is DMA, flow control must still be provided for.)

Should Hosts require an OPE to be attached via a Link Level that furnishes physical demultiplexing (e.g., a group of RS232 ports), any attempt to avoid furnishing reliability and explicit flow control, is done at their peril; we have not chosen to assist such an enterprise, but neither have we precluded it. (It would certainly violate the spirit of the thing, however.)

Command Level Interaction

The approach chosen for this H-FP is to have the interaction of a small set of commands, separately applicable to a given Channel Level channel. The commands are simple, but sufficiently flexible to permit the off-loading of the interpreters of the large number of protocols at various levels in the hierarchy. This flexibility is made possible in part by the similar nature of the interfaces to most protocols, combined with the provision of "protocol idiosyncratic parameters". These parameters are defined for each offloaded protocol interpreter in the OPE. The use of such parameters does not complicate the basic design of the OPE, since it must be customized for each off-loaded protocol anyway, and all that is required of the OPE for those parameters is to pass them to the off-loaded protocol interpreter. Hence, an interface tailored to a particular protocol can be created in a straightforward and cost-effective way.

The command dialog is more or less asynchronous. Commands can be issued at any particular time (except when there is a pending command, which will be discussed below), and there is no need for dummy traffic on a channel when no commands are issued.

Associated with each command is a response. The purpose of this

response is to indicate, at some level that depends in part on the particular protocol interpreter that is offloaded to the OPE, whether the command was successfully executed, and if unsuccessful, the reason. Often, generating the response involves interaction with the protocol interpreter before a response can be generated.

When a command is issued, the issuer must wait for a response before another command is issued. The nature of the communication between the Host and the OPE is thus a lock step command/response dialog. There are two major exceptions to this principle, however. One exception is the abrupt form of the End command, which can be issued at any time to cancel any previously issued commands, and indicate that services are no longer desired. The other exception is the Signal command. Since a Signal is out-of-band and usually of high importance, forcing it to wait on a response would be undesirable. Hence, a Signal command can be issued while commands (other than Signal) are pending. However, a Signal command should not be issued before a successful response to the Begin command has been received. Since it is possible for more than one command of different types to be pending at one time, a mechanism to distinguish responses is needed. Since there are never two commands of the same type pending, including the command name in the response is sufficient to make this distinction.

A special case command is the Transmit command. Details of the Transmit command are provided in the next section. Essentially, the Transmit command is used to invoke the data transfer services of the off-loaded protocol (when issued by the Host) or to indicate the arrival of new data from the network (when issued by the OPE). The nature of specific protocol interfaces for these events varies widely between protocols. Some may block until the data is accepted by the remote counterpart (or "peer") protocol interpreter, while others may not. Hence, there is a special parameter which indicates the nature of the Transmit command interface. It can either require that the response should be generated immediately after determining the Transmit command is complete and formed properly, or can indicate that the response should not be generated until the appropriate interface event is given by the remote protocol interpreter. The default action for all Transmit commands can be initialized using the Begin command and changed using the Condition command. Also, the default action can be temporarily overridden by specifying a parameter with the Transmit command. The net result of this mechanism is to allow the Host to determine within reason just how lock-stepped transmissions are to be. (It is assumed that the usual case will be to transfer the burden of buffering to the OPE by taking immediate responses, provided that doing so "makes sense" with the particular offloaded protocol in play.)

RFC 929

December 1984

Proposed Host-Front End Protocol

Some protocols provide a block-oriented data transfer service rather than a stream-oriented one. With such a service, the data associated with a transfer request is viewed as an integral unit. For actual network transmission, the protocol may permit these units to be grouped or fragmented. However, the receiving end must deliver the data in the original, integral units. Protocols that conform to this model include some datagram protocols such as IP and UDP, and also some connection protocols such as NBS TP.

To cater to these types of protocols, it is a convention that commands, their parameters, and any associated data be transferred between the Host and the OPE in a single chunk. Any data associated with an H-FP command is viewed as an integral unit which is used in the corresponding service request given to the outboard protocol interpreter or delivered as a complete unit to the process in the Host. Operation of stream-oriented protocols such as TCP will not be adversely affected by this convention.

To accommodate Channel protocols that do not provide for arbitrarily large chunks, a mechanism at the Command level is required to permit the linking of multiple chunks into a single command, in order to transfer the burden of buffering as much as possible from the Host to the OPE. The facility proposed here would consist of an indication at the beginning of each chunk which would distinguish integral commands, fragments of a command for which more fragments are yet to arrive, and the final fragment of a command. The details of this mechanism are discussed in the section on the syntax of commands and responses.

It is a convention for this H-FP that any data associated with a command must start on a word boundary (as defined by the local system). Consequently, there is a need to provide padding within the commands. Such padding is used only to fill to the next appropriate boundary, and has no semantic significance to the command interpreter (i.e., two commands that are identical except for the amount of padding should behave identically). The details of this padding are discussed in the section on the syntax of commands and responses.

RFC 929
Proposed Host-Front End Protocol

December 1984

Syntax Rules

At the Command Level, communication between the Host and the OPE takes the form of commands and responses. A command is a request for some particular action, and the response indicates the success or failure of performing the requested action.

All commands and responses are coded in ASCII characters. (Nothing precludes OPEs from accepting EBCDIC from Hosts that use it in native mode, but that is not required.) These characters are sent in some way convenient for the Host, and the OPE is sufficiently flexible to interpret them. (i.e., OPEs are expected to accommodate Host idiosyncracies in regard to such things as use of 7-bit ASCII in a 9-bit field.) This approach offers several advantages:

Adaptabilities in most Hosts: Most Hosts have the ability to generate and interpret ASCII character streams. Hence, integrating H-FP into a Host will not require difficult software.

Script generation: Generation of test and operational command scripts will be simplified, since they will not need to contain special characters.

Terminal Operation: Using simple command streams simplifies the conversion of an OPE to a generic virtual terminal support machine. This is particularly useful during development and testing.

Testing: Testing will not require special hardware to interpret commands and responses. A terminal or data line analyzer would be adequate.

The specific format for the commands and responses will be discussed in the sections that follow. In those sections, the quote character is used to indicate strings. The symbols "<" and ">" (referred to as angle brackets) are used as meta-characters.

Syntax of Commands

As alluded to in the section discussing the interaction discipline between the Host and the OPE, a function is provided by which a chunk can be used to carry either a complete command or a fragment of a command. The mechanism chosen to provide this function entails use of the first character position in the chunk as a chunk usage identifier. The character "C" in the first position indicates a chunk containing a single, complete command. "F" in the first position indicates a chunk which is the first part of a multichunk command. "M" in the first position indicates the chunk is a middle

Lillenkamp & Mandell & Padlipsky

[Page 7]

RFC 929
Proposed Host-Front End Protocol

December 1984

part (neither the first nor the last chunk) of a command. Finally, "L" indicates the chunk is the last chunk of a multi-chunk command. Hence, the following sequences of chunks (the letter corresponds to the chunk usage identifier in each chunk, and the angle brackets enclose a chunk) are legal:

```
<C>
<F><L>
<F><M><M><L>
```

while the following are not legal:

```
<L>
<M><L>
<F><C>
```

Tactics for handling multiple chunks with regard to OPE buffering limits are left to the ingenuity of OPE builders. The spirit is to take as much as you can, in order to relieve the Host of the necessity of buffering itself.

A command always begins immediately following the indicator character, with possible intervening spaces. This implies a chunk can contain at most one complete command. The end of the command (not including the data) is signified by a newline (denoted as <nl> in this document) that does not appear inside a quoted string (see below). The end of the data is designated by the end of the last chunk.

Commands take the form of an ASCII string. The command identifier is the first word of the chunk. It consists of at least the first two letters of the command, in either upper or lower case (e.g., the sequences "BE", "Be", "bE", and "be" all identify the Begin command). Additional letters of the command name can be included if desired to aid readability of the command stream.

Following the command identifier is a list of parameters. These parameters are also represented as ASCII strings, although the specific format will depend on the particular parameter. The data to be transmitted is not considered a control parameter, however, and need not be ASCII data.

Parameters are separated by one or more spaces. Tabs, newlines, and other white space are not legal parameter separators.

Parameter strings may be quoted, using the character "<". Any

RFC 929
Proposed Host-Front End Protocol

December 1984

characters between the <"> characters are a part of the parameter, including spaces and newlines. The character <"> that is part of the parameter is represented inside a quoted string as <"">.

The order in which the parameters appear within the command is significant to their interpretation by the Host and by the OPE. Optional parameters may be skipped by using the characters ".,." to indicate a NULL parameter. Such a NULL parameter takes its default value. Alternatively, each parameter has a MULTICS/UNIX style Control Argument/Flag associated with it that can be used to identify the parameter, without placing NULL parameters for each parameter skipped. This flag consists of one or two ASCII characters, and either upper or lower case may be used. For example, if the fourth parameter of a command had a flag of "-p" and the user wished the first three parameters to be null, he could use:

command -p value

or

command -P value

instead of

command value

if it were more convenient for the Host to do so. Flagged parameters must still appear in the correct sequence within the command, however.

There may be data associated with some of the commands. Any such data is placed into the chunk following all the parameters and the unquoted newline. Padding can be provided by placing spaces between the end of the final parameter string and the newline, so that data begins on a word boundary. The OPE will always pad to a host word boundary. Padding by hosts is optional.

Syntax of Responses

Responses are actually just a special form of a command. It is anticipated that all responses would fit into a single channel chunk, although the mechanisms described for multichunk commands can certainly be used in responses. The ASCII string used to uniquely identify the response command is "RE" ("Re", "rE", and "re" are also permitted).

After the response command identifier is the original command

RFC 929
Proposed Host-Front End Protocol

December 1984

identifier, so the response can be associated with the proper command. Following this identifier is a three ASCII digit response code, a set of protocol idiosyncratic parameters, and a textual message. The protocol idiosyncratic parameters are used to transfer interface information between the Host and the OPE, and may not be needed when off-loading some protocol interpreters. The textual message is intended for human interpretation of the response codes, and is not required by the protocol. The three digits uniquely identify the semantics of the response, at least within the context of a particular command and particular outboarded protocol interpreter.

Responses are numerically grouped by the type of information they convey. The first digit identifies this group, and the last two digits further qualify the reply. The following list illustrates this grouping.

- 0XX Successful: The command was executed successfully. The response code may contain further information.
- 1XX Conditional Success: The command was executed successfully, but not exactly according to the service and flow control suggestions. If those suggestions were particularly important to the requester, he may wish to issue an End command. The response code contains information on what suggestion or suggestions could not be followed.
- 2XX Command Level Error: An error at the command level has occurred. This could include requesting services of a protocol not supported, or a problem in the way those services were requested. This level does not include problems with the syntax of the command or its parameters.
- 3XX Syntax and Parameter Errors: An error in the syntax of the command or a problem with one of its parameters has occurred. A problem with a parameter may be other than syntactical, such as illegal address.
- 4XX Off-loaded Protocol Interpreter Problems: Some problem with the particular off-loaded protocol has occurred.
- 5XX Local OPE Internal Problems: Problems, such as insufficient OPE resources, or problems with OPE to subnet interface.
- 6XX Security Problem: Some problem with Host, network, or OPE security has occurred. The response code indicates the problem.

REC 929

December 1984

Proposed Host-Front End Protocol

7XX Reserved for Future Expansion

8XX Reserved for Future Expansion

9XX Protocol Idiosyncratic Errors: Some error occurred that is idiosyncratic to the particular off-loaded protocol being used. The response code indicates the error.

Description of the Commands

As stated above, communication between the Host and the OPE at the Command Level is accomplished using commands and responses. Commands may be issued by either the Host or the OPE, and are used to stimulate activity in the other entity. Some commands may only have a meaningful interpretation in one direction, however. A response indicates that the activity started by the command was completed, and a code indicates success or failure of the command, and perhaps other information related to the command as well.

Associated with each command is a set of parameters. The order in which the parameters appear is significant to the correct operation of the protocols. More information on the syntax of command parameters can be found in the syntax descriptions.

The commands are:

- Begin: initiate communication between a process in the Host and an off-loaded protocol interpreter in the OPE. (A Channel level stream/connection will typically have been opened as a prior step. All other commands, except No-op, apply to a stream on which a successful Begin has been done.)
- Transmit: transmit data between a process in the Host and an off-loaded protocol interpreter in the OPE.
- Signal: cause an out-of-band signal to be sent by the off-loaded protocol interpreter to its peer, or indicate the arrival of such a signal from the remote side.
- Condition: alter the off-loaded protocol interpreter's operational characteristics.
- Status: transfer status requests or information between a process in the Host and an off-loaded protocol interpreter in the OPE.

RFC 929

December 1984

Proposed Host-Front End Protocol

- End: indicate that services from the off-loaded protocol interpreter are no longer required, or will no longer be provided.
- No-op: performs no operation, but facilitates testing.

These commands will be discussed in the following sections. Each of these sections includes a discussion of the purpose of the command, a description of each of the parameters used with the command, a list of responses for the command, an example of the command, and a set of notes for the implementor. (An appendix will eventually be furnished for each protocol offloading, showing the use of its protocol idiosyncratic parameters as well as of the general parameters on a per-command basis. Initially, only representative offloadings will be treated in appendices, with others to be added after the protocol gains acceptance.)

Begin

Purpose of the Begin Command

The purpose of a Begin command is to initiate communication between the Host and the OPE on a particular stream or channel (the channel is opened as a separate step, of course). The interpretation of the command is somewhat dependent upon whether it was issued by the Host or the OPE.

- If the command was issued by the Host, it means some process in the Host is requesting services of a protocol that was off-loaded to the OPE. The user request results in the establishment of a channel connection between the Host and the OPE, and a Begin command to the Command interpreter in the OPE.
- If the command was issued by the OPE, it means some protocol interpreter in the OPE has data for some process in the Host which is not currently known by the OPE. An example would be an incoming UDP datagram on a new port, or if no Begin for UDP had been issued at all by the Host. (An incoming TCP connection request could be handled by a response to the user's Passive Open request, which had previously caused a Begin request from the Host; an incoming TCP connection request to a port on which no Listen had been issued would cause an OPE generated Begin, however.)

As indicated earlier, any particular Host is not required to support two-way Begins.

RFC 929
Proposed Host-Front End Protocol

December 1984

Parameters of the Begin Command

The Begin command has several parameters associated with it. These parameters contain information needed by the offloaded protocol to provide an adequate level of network service. This information includes protocol, source and destination addresses, and also type of service and flow control advice. These parameters are discussed in detail below.

Protocol

The protocol parameter identifies that off-loaded protocol in the OPE to which Begin is directed, or which issued the Begin to the Host. For example, if the user wished to utilize TCP services, and the TCP software was off-loaded into the OPE, then the Protocol parameter for the Begin command would be TCP.

There are two categories of protocol parameters -- generic and specific. A generic parameter identifies a type of protocol service required, but does not identify the actual protocol. Use of generic protocols allows a Host process to obtain network services without specific knowledge of what protocol is being used; this could be appropriate for use in situations where no specific aspect(s) of a specific protocol is/are required. For example, the user may select a generic Host-to-Host connection protocol, and (at some point in the future) may actually receive services from either TCP or the NBS Transport Protocol, depending on the network (or even the foreign Host) in question. A specific protocol parameter identifies some particular protocol, e.g., TCP, whose use is required for the given channel.

The valid entries for the protocol field include:

Generic	Specific	Comment
GIP	IP	Datagram Internetwork Protocol
HHP	TCP	Connection Transport/Host-Host Protocol
GDP	UDP	Datagram Transport/Host-Host Protocol
VTP	TEL	Virtual Terminal (Telnet) Protocol
GFP	FTP	File Transfer Protocol
MAIL	SMTP	Mail Transfer Protocol
PROX	PROX	Proximate Net Interface Protocol

(Note that the final line is meant to allow for a process in an OPE'd Host's getting at the PI of the Network Interface Protocol for whatever the proximate network is. Of course, so

RFC 929

December 1984

Proposed Host-Front End Protocol

doing only makes sense in specialized contexts. We conceive of the desirability of "pumping bits at a peripheral" on a LAN, though, and don't want to preclude it, even if it would be impossible on many LAN's to deal with the problem of distinguishing traffic coming back on the LAN in this "raw" mode from normal, IP traffic. Indeed, in some contexts it is likely that administrative considerations would preclude avoidance of IP even if technical considerations allowed it, but it's still the case that "the protocol" should provide a hook for going directly to the L I protocol in play.)

There is no default value for this parameter. If it is not present, the Begin command is in error. The control flag for this parameter is -pr.

Active/Passive

The Active/Passive parameter indicates whether the issuer of the Begin command desires to be the Active or Passive user of the protocol. This parameter is particularly relevant to connection-oriented protocols such as TCP, where the user may actively pursue connection establishment, or else may passively wait for the remote entity to actively establish the connection; it also allows some process to establish itself as the Host "fielder" of incoming traffic for a connectionless protocol such as IP.

Active is requested using the single character "A". Passive is indicated using the character "P". The default value of this parameter is "A". Also, when the OPE issues the Begin command, the value must be "A". The control flag for this parameter is -ap.

Foreign Address Primary Component

The addressing structure supported by H-FP is two level. Each address has two components, the primary and the secondary. The exact interpretation of these two components is protocol specific, but some generalities do apply. The primary component of the address identifies where the protocol is to deliver the information. The secondary component identifies which recipient at that location is to receive the information. For example, the TCP primary address component is the Host's Internet Address, while the secondary address component is the TCP port. Similarly, IP's primary address component is the Host's Internet Address, and the secondary address component is the IP ULP field. Some protocols provide only a single level

of addressing, or the secondary level can be deduced from some other information (e.g., Telnet). In these cases, only the primary component is used. To cater to such cases, the secondary component parameter comes later in the parameter list.

The Foreign Address Primary Component parameter contains the primary component of the destination address. It may be in either a numeric or symbolic form. (Note that this allows for the OPE to exercise a Name Server type of protocol if appropriate, as well as freeing the Host from the necessity of maintaining an in-board name to address table.) The default value for this parameter, although it only makes sense for Passive Begins, is "Any Host". The control flag for this parameter is -fp.

Mediation Level

The mediation level parameter is an indication of the role the Host wishes the OPE to play in the operation of the protocol. The extreme ranges of this mediation would be the case where the Host wished to remain completely uninvolved, and the case where the Host wished to make every possible decision. The specific interpretation of this parameter is dependent upon the particular off-loaded protocol.

The concept of mediation level can best be clarified by means of example. A full inboard implementation of the Telnet protocol places several responsibilities on the Host. These responsibilities include negotiation and provision of protocol options, translation between local and network character codes and formats, and monitoring the well-known socket for incoming connection requests. The mediation level indicates whether these responsibilities are assigned to the Host or to the OPE when the Telnet implementation is outboard. If no OPE mediation is selected, the Host is involved with all negotiation of the Telnet options, and all format conversions. With full OPE mediation, all option negotiation and all format conversions are performed by the OPE. An intermediate level of mediation might have ordinary option negotiation, format conversion, and socket monitoring done in the OPE, while options not known to the OPE are handled by the Host.

The parameter is represented with a single ASCII digit. The value 9 represents full OPE mediation, and the value 0 represents no OPE mediation. Other values may be defined for

RFC 929

December 1984

Proposed Host-Front End Protocol

some protocols (e.g., the intermediate mediation level discussed above for Telnet). The default value for this parameter is 9. The control flag for this parameter is -m.

Transmit Response Discipline

The Transmit Response Discipline parameter is used to set the desired action on the OPE's part for generating responses to Transmit commands. Essentially the parameter determines when the OPE's response to the transmit command occurs (i.e., immediately or delayed).

The Transmit Response Discipline value is represented by a single ASCII character. The character "N" is used for nonblocking Transmit commands, which implies that responses for Transmit commands should be generated as soon as the command has been examined for correctness (i.e., that the syntax is good and the parameters appear reasonable). In other words, the outboard protocol interpreter has the data in its queue, but hasn't necessarily transmitted it to the net. The character "B" is used for blocking Transmit commands, which requests that the response not be generated until the protocol interpreter has successfully transmitted the data (unless, of course, the Transmit command was badly formed). The default value for this parameter is "N", or a nonblocking Transmit command. The control flag for this parameter is -tr. (Depending on the protocol in play, "successfully transmitted" might well imply that an acknowledgment of some sort has been received from the foreign Host, but for other protocols it might only mean that the given collection of bits has been passed from the OPE to the proximate net.)

Foreign Address Secondary Component

The addressing mechanisms supported by this level of H-FP are discussed above. The Foreign Address Secondary Component parameter contains the value of the destination address's secondary component. Some protocols do not require this parameter, or can obtain it from other information. Therefore, the default value for this parameter is NULL. A NULL secondary component might be an error for some protocols, however. The secondary component can be expressed either numerically or symbolically. The control flag for this parameter is -fs. (Note that it is intended to be "legal" to specify a Secondary Component other than the Well-Known Socket for the protocol in play; in such cases, the result should be that the virtualizing of the given protocol be applied to the stream, in the

expectation that that's what the other side is expecting. This is to cater to, for example, a Terminal-Terminal protocol that merely "does Telnet" to a socket other than the usual Logger.)

Local Address Secondary Component

The Local Address Secondary Component parameter contains the value of the local address's secondary component. (The primary component is assumed to be the default for the Host, but can be altered as well; see below.) Some protocols do not require this parameter, or can obtain it from other information. In some cases, the OPE may already know the value for this parameter and therefore not require it. The default value of this parameter is NULL. The local address secondary component can be expressed either numerically or symbolically. The control flag for this parameter is -ls.

Begin Timeout Interval

After a Begin command is issued, a timer can be started. If the activity requested cannot be performed within some timed interval, then the Begin command may expire. An expired Begin command returns a response code indicating a Begin timeout occurred. The Begin Timeout Interval parameter contains the length of time the timer will run before the Begin timeout occurs.

The parameter is represented as a string of ASCII digits indicating the time interval in seconds. The default value of this parameter is infinity (i.e., the Begin command will never timeout). The control flag for this parameter is -bt.

Type of Service Advice

The Type of Service Advice parameter contains information on the service characteristics the user desires from the offloaded protocol. Included in this parameter is the precedence of the data transfer, and also indication of whether high throughput, fast response time, or low error rate is the primary goal.

The format of this parameter is a letter immediately (i.e. no intervening spaces) followed by a digit. The letter "T" indicates that high throughput is desired. The letter "R" indicates minimal response time is the goal. The letter "E" indicates that low error rates are the goal. The letter "N" indicates there are no special service requirements to be conveyed. The digit immediately following the character

RFC 929
Proposed Host-Front End Protocol

December 1984

indicates the desired precedence level, with zero being the lowest, and nine being the highest. The specific interpretation of this parameter is dependent on what service options are provided by the protocol. The default value of this parameter is the lowest precedence (ROUTINE), and no special service requests. The control flag for this parameter is -ts.

Flow Control Advice

The Flow Control Advice parameter contains information on the flow characteristics desired by the user. Some applications such as file transfer operate more efficiently if the data is transferred in large pieces, while other, more interactive applications are more efficiently served if smaller pieces are used. This parameter then indicates whether large or small data blocks should be used. It is only relevant in stream or connection-oriented protocols, where the user sends more than a single piece of data.

This parameter is represented by a single ASCII digit. A value 0 means the data should be sent in relatively small blocks (e.g., character or line oriented applications), while a value 9 means the data should be sent in relatively large blocks (e.g., block or file oriented applications). Other values represent sizes between those extremes. The character "N" indicates that no special flow control advice is provided. The actual interpretation of this parameter is dependent on the particular protocol in the OPE. The default value of this parameter is no flow control advice. In this case, the protocol in the OPE will operate based only on information available in the OPE. The control flag for this parameter is -fc.

Local Address Primary Component

This parameter contains the local address primary component. It is anticipated that under most circumstances, this component is known to both the Host and the OPE. Consequently, this parameter is seldom required. It would be useful if the Host desired to select one of several valid addresses, however. The control flag for this parameter is -lp.

Security

The security parameters contain a set of security level, compartment, community of interest, and handling restriction information. Currently, security is provided by performing all

RFC 929
Proposed Host-Front End Protocol

December 1984

processing at system high level or at a single level. Consequently, these parameters are probably redundant, since the security information is known. In the future, however, these parameters may be required. Therefore a field is provided. The control flag for this parameter is -s.

Protocol Idiosyncratic Parameters

The remaining parameters are protocol idiosyncratic. That is, each protocol that is off-loaded may have a set of these parameters, which are documented with a description of the off-loaded protocol. The default value for these parameters is NULL, unless otherwise specified by a particular offloaded protocol. The control flag for this set of parameters is -pi, which identifies the first protocol idiosyncratic parameters. Control flags for other protocol idiosyncratic parameters must be defined for each off-loaded protocol.

Data

After the Protocol Idiosyncratic Parameters, if any, and the required <nl>, if the protocol in play allows for it at this juncture the rest of the chunk will be interpreted as data to be transmitted. That is, in connection oriented protocols data may or may not be permitted at connection initiation time, but in connectionless protocols it certainly makes sense to allow the H-FP Begin command to convey data. (This will also be useful when we get to the Condition command.)

Responses

The following responses have been identified for the Begin command:

000	Command completed successfully
101	Throughput not available; using maximum
102	Reliability not available; using maximum
103	Delay not available; using minimum
110	Flow Control advice not followed; smaller blocks used
111	Flow Control advice not followed; larger blocks used
201	Failed; Begin not implemented in this direction
202	Failed; timeout
203	Failed; Begin command on already active channel
300	Problem with multiple chunks
301	Syntax problem with Begin command
302	Protocol not supported in OPE/Host
303	Active service not available

Lillienkamp & Mandell & Padlipsky

[Page 19]

RFC 929
Proposed Host-Front End Protocol

December 1984

304	Passive service not available
305	Invalid Foreign Address Primary Component
306	Invalid Transmit Discipline
307	Invalid Foreign Address Secondary Component
308	Invalid Local Address Secondary Component
309	Invalid Timeout Interval
310	Invalid Type of Service Advice
311	Invalid Flow control Advice
312	Invalid Local Address Primary Component
401	Protocol Interpreter in OPE not responding
402	Remote Protocol Interpreter not available
403	Failed; insufficient protocol interpreter resources
501	Failed; insufficient OPE resources
601	Request violates security policy
602	Security parameter problem

Additionally, protocol idiosyncratic responses will be defined for each off-loaded protocol.

Example of Begin Command

The Begin command is the most complex of the H-FP Command Level. When the off-loaded protocol is TCP, the Begin command is used to open TCP connections. One possible example of a Begin command issued by an inboard Telnet interpreter to open a TCP connection to ISIA, with no begin timeout interval, is:

```
C BE TCP A ISIA 9 N 23 . . . NO S <nl>
```

Where:

TCP	The code for the protocol TCP
A	Indicates Active Begin
ISIA	The name of a Host at USC-ISI
9	Mediation Level 9: Full OPE mediation
N	Non-blocking transmit
23	Destination Telnet Port
. .	skip over parameters (Local Address Secondary, Begin Timeout Interval)
NO	Type of Service Advice: No special Advice, Normal Precedence
S	Flow Control Advice: use small blocks

This command will cause the OPE to invoke the TCP interpreter to generate the initial SYN packet to the well-known Telnet socket on Host ISIA. It also informs the OPE to do all TCP related processing via the Mediation Level, accepts default

RFC 929
Proposed Host-Front End Protocol

December 1984

Local Address parameters, and sets the Begin Timeout Interval to infinity. The precedence of the TCP connection is Normal, and the TCP interpreter is informed that the data stream will consist of primarily small blocks.

Notes to the Implementor

Response 203 might seem silly to some readers, but it's there in case somebody goofed in using the Channel Layer.

Transmit

Purpose of the Transmit Command

The purpose of the Transmit command is to permit the process in the Host to send data using an off-loaded protocol interpreter in the OPE, and also to permit the OPE to deliver data received from the network destined for the process in the Host. The Transmit command is particularly relevant to connection and stream type protocols, although it has applications for connectionless protocols as well. After the Begin command is issued successfully and the proper Response received, Transmit commands can be issued on the given channel. The semantics of the transmit command depend on whether it was issued by the Host or the OPE.

- If the Host issues the Transmit command, a process in the Host wishes to send the data to the destination specified to the off-loaded protocol interpreter that was established (typically) by a previous Begin command on the given H-FP channel.

- If the OPE issues the command, the OPE has received data destined for a process in the Host from a connection or stream supported by the off-loaded protocol that was established by a previous Begin command on the given H-FP channel.

Parameters of the Transmit Command

The Transmit command has one parameter associated with it. It is an optional parameter, to temporarily override the response discipline for this particular transmit command. Some protocols may have protocol-idiosyncratic parameters as well. The transmit command also has data associated with it. All parameters must precede the data to be transmitted.

RFC 929

December 1984

Proposed Host-Front End Protocol

Response Discipline Override

The Response Discipline Override parameter indicates the desired response discipline for that individual Transmit Command, overriding the default response discipline. A single ASCII character is used to indicate the desired discipline. The character "N" indicates that this Transmit command should not block, and should return a response as soon as the data is given to the protocol interpreter in the OPE. The character "B" indicates that this Transmit command should block, meaning that a response should not be generated until the data has been sent to the destination. The default value of this parameter is the currently defined Transmit Command response discipline. The use of this parameter does not alter the currently defined Transmit command response discipline; the default is changed with the Condition command. The control flag for this parameter is -rd.

Protocol-Idiosyncratic Parameters

Any other parameters to the Transmit command are protocol-idiosyncratic. That is, each protocol that is off-loaded has a set of these parameters, which are documented with a description of the off-loaded protocol. The default value for these parameters is NULL, unless otherwise specified by a particular off-loaded protocol. The control flag for this set of parameters is -pi, which identifies the first protocol-idiosyncratic parameters. Control flags for other protocol-idiosyncratic parameters must be defined for each off-loaded protocol.

Responses

The following responses for the Transmit command have been identified:

000	Transmit Command completed successfully
201	Transmit Command not appropriate
300	Problem with multiple chunks
301	Syntax problem with Transmit Command
302	Invalid Transmit Command Response Discipline
401	Protocol Interpreter in OPE not responding
402	Failure in remote protocol interpreter
403	Failed; insufficient protocol interpreter resources
501	Failed; insufficient OPE resources
601	Request violates security policy

RFC 929
Proposed Host-Front End Protocol

December 1984

Additionally, protocol-idiosyncratic responses will be defined for each off-loaded protocol.

Example of Transmit Command

The transmit command is used in TCP to provide the TCP write call. An example of such a transmit command would be:

```
C TR N <nl> <DATA>
```

Where N indicates non-blocking transmission discipline, <nl> is the required command-ending newline, and <DATA> is presumed to be the user's data that is to be transmitted.

Notes to the Implementor

If you get a 403 or a 501 response and have sent a multiple chunk it probably makes sense to try a single chunk; if you've sent a single chunk, it makes sense to wait a while and try again a few times before giving up on the stream/channel.

Condition

Purpose of the Condition Command

The primary purpose of the Condition command is to permit a process to alter the characteristics that were originally set up with the Begin command. (That is, "condition" is a verb.) These characteristics include the addresses, the mediation level, the type of service, and the flow control parameters from Begin. They may also include protocol-idiosyncratic characteristics. (Although Condition is usually thought of as a Host->OPE command, it may also be used OPE->Host in some contexts.)

Condition is a generic command that may find little use in some off-loaded protocols. In others, only some of the parameters identified may make sense. For example, changing the destination address of a TCP connection involves closing one connection and opening another. Consequently, it may make more sense to first issue an End command, and then a Begin with the new address. In other protocols, such as IP or UDP, changing the address on each datagram would be a perfectly reasonable thing to do.

RFC 929
Proposed Host-Front End Protocol

December 1984

Parameters of the Condition Command

The Condition command has the same parameters as the Begin command. Any parameters expressed in a Condition command indicate the new values of the characteristics to be altered; all parameters not expressed retain the current value.

Although it is possible to express the change of any of the characteristics originally set up in the Begin command using the Condition command, there are some characteristics that do not make sense to alter, at least for some protocols. For example, once a connection is opened, it does not make much sense to change the Foreign Address Primary or Secondary Components. Doing so is inconsistent with current versions of TCP, and would require the closing of the existing connection and opening a new one to another address. Earlier versions of TCP did permit connections to be moved. If a protocol that provided such a feature was implemented in the OPE, the changing the Secondary Address Components would be a reasonable thing to do.

Responses

The responses to the Condition command are the same as those to the Begin command.

Example of Condition Command

The Condition Command can be quite complex, and can be used for many purposes. One conceived use of the condition command would be to change the type of service advice associated with the channel. An example of this (which also demonstrates the ability to skip parameters) is:

```
C -ts T <nl>
```

which causes the offloaded PI associated with the current channel to attempt to achieve high throughput (in its use of the comm subnet(s) in play).

Notes to the Implementor

Signal

Purpose of Signal Command

The purpose of the Signal Command (implicitly at least) is to permit the transfer of out-of-band signals or information between the Host and the OPE, in order to utilize (explicitly) out-of-band signaling services of the off-loaded protocol. The semantics of the Signal command depend upon whether it was issued by the Host or the OPE.

- If the Signal command was issued by the Host, it means a process in the Host desires to send out-of-band data or an out-of-band signal.

- If the Signal command was issued by the OPE, it means out-of-band data or an out-of-band signal arrived for the process associated with the channel in the Host.

Parameters of the Signal Command

The basic usage of the Signal command is with no parameters, which sends or reports the receipt of an out-of-band signal. Some protocols, such as the NBS Transport Protocol, permit the user to send data with the out-of-band signal. Hence, data is permitted to accompany the Signal command. There may also be protocol-idiosyncratic parameters for the Signal command. If this is the case, these parameters would come before the data.

Protocol-Idiosyncratic Parameters

The parameters for the Signal command are protocol idiosyncratic. That is, each protocol off-loaded has a set of these parameters. The default value for these parameters is their previous values. Control flags for multiple protocol-idiosyncratic parameters must be defined for each off-loaded protocol.

Responses

The following responses have been identified for the Signal command:

000	Command completed successfully
201	Command not appropriate
300	Problem with multiple chunks
301	Syntax problem with Command

RFC 929

December 1984

Proposed Host-Front End Protocol

- 401 Protocol Interpreter in OPE not responding
- 402 Failure in remote protocol interpreter
- 403 Failed; insufficient protocol interpreter resources
- 501 Failed; insufficient OPE resources
- 601 Request violates security policy

Additionally, protocol-idiosyncratic responses will be defined for each off-loaded protocol.

Example of Signal Command

The major perceived use for the Signal command when offloading a connection protocol is sending an out-of-band signal with no data. In such a case, the appropriate signal command would be:

```
C SI <nl>
```

Notes to the Implementor

Some protocols may allow only one outstanding signal at a time. For these protocols, it is an implementation issue whether the OPE will buffer several signals, but a good case could be made for the position that a scrupulous OPE would reflect a 202 response back to the Host in such cases.

There is some question as to the proper handling of the "expedited data" notion of some (particularly ISO) protocols. It might be more appropriate to deal with such a thing as a protocol idiosyncratic parameter on the Transmit command instead of using the Signal command (even if it's the closest approximation to an out-of-band signal in the given protocol). If it's provided using the Signal command, the expedited data should not be passed as ASCII, and should appear after the command-terminating newline character (and appropriate padding with space characters).

Status

Purpose of Status Command

The purpose of the Status command is to permit the Host to request and obtain status information from the OPE, and vice versa. This includes status request of a conventional protocol interface (e.g., in TCP, there is a request to determine the state of a particular connection).

RFC 929
Proposed Host-Front End Protocol

December 1984

Parameters of the Status Command

The parameters for the Status command indicate whether it is a request or a response, and contain the status information.

Request/Report

This parameter indicates whether the command is a Status request or a Status report. It consists of a single ASCII character. Q indicates a request (query), and R indicates a report. It should be noted that a report may be generated as the result of a query, or may be generated as the result of specific protocol mechanisms.

Protocol-Idiosyncratic Parameters

The parameters to the status command are protocol-idiosyncratic. That is, each protocol off-loaded has a set of these parameters. The default value for these parameters is their previous values. Among these parameters is an identifier of the type of status information contained or requested, and a value or set of values that contain the particular status information. The status information itself should be the last item in the command. The control flag for this set of parameters is -pi, which identifies the first protocol-idiosyncratic parameters. Control flags for other protocol-idiosyncratic parameters must be defined for each off-loaded protocol.

Responses

The following responses have been identified for the Status command:

000	Command completed successfully
201	Command not appropriate
300	Problem with multiple chunks
301	Syntax problem with Command
302	Inappropriate status request
303	Inappropriate status response
401	Protocol Interpreter in OPE not responding
402	Failure in remote protocol interpreter
403	Failed; insufficient protocol interpreter resources
501	Failed; insufficient OPE resources
601	Request violates security policy
9xx	Protocol Idiosyncratic status responses

Lilienkamp & Mandell & Padlipsky

[Page 27]

RFC 929
Proposed Host-Front End Protocol

December 1984

Example of Status Command

The status command can be particularly complex, depending on the protocol and particular type of status information. One possible use of the status command when off-loading TCP is to communicate the status service request. For performing this operation the status command would be:

C ST Q <nl>

Notes to the Implementor

End

Purpose of the End Command

The purpose of the End command is to communicate that services of the off-loaded protocol are not required. The semantics of the End command depends upon whether it was issued by the Host or the OPE.

- If the Host issues the End command, it means the process in the Host no longer requires the services of the offloaded protocol.
- If the OPE issues the End command, it means the remote entity has no more data to send (e.g., the off-loaded protocol is TCP and the remote user has issued a TCP close).

Parameters of the End Command

One parameter is associated with the End Command. It indicates whether the termination should be "graceful" or "abrupt" (see below).

Graceful/Abrupt

The Graceful/Abrupt parameter indicates whether the End should be handled gracefully or abruptly. If it is handled gracefully, then data in transit is allowed to reach its destination before service is actually terminated. An abrupt End occurs immediately; all data transmitted from the Host but still pending in the OPE is discarded, and no new incoming data is sent to the Host from the OPE.

RFC 929
Proposed Host-Front End Protocol

December 1984

The parameter is indicated by a single ASCII character. The character "G" denotes graceful, and "A" denotes abrupt. The default value for this parameter is graceful.

Responses

The following responses have been identified for the End command:

000	Command completed successfully
201	Command not appropriate
300	Problem with multiple chunks
301	Syntax problem with Command
302	Illegal Type of End Command
401	Protocol Interpreter in OPE not responding
402	Failure in remote protocol interpreter
403	Failed; insufficient protocol interpreter resources
501	Failed; insufficient OPE resources
601	Request violates security policy

Additionally, protocol idiosyncratic responses will be defined for each off-loaded protocol.

Example of End Command

The syntax of the End command is relatively straightforward. It consists of a chunk that contains only a chunk usage identifier, the end command string, and the parameter indicating whether the end should be graceful or abrupt. A possible valid (abrupt) End command would be:

C EN A <nl>

Notes to the Implementor

Once an End has been issued in a given direction any other commands on the channel in the same direction are in error and should be responded to appropriately.

RFC 929
Proposed Host-Front End Protocol

December 1984

No-op

Purpose of the No-op Command

The No-op command performs no operation. Its purpose is to permit the Host and OPE to participate in a dialog which does not alter the state of communication activities, both for debugging purposes and to support features of certain protocols (e.g., Telnet's Are You There command).

Parameters of the No-op Command

There are no parameters associated with the No-op command.

Responses

There are only two possible legal responses to the No-op command. They are:

000	No-op Command Completed Correctly
300	Problem with multiple chunks

Example of No-op Command

Syntactically the No-op command is quite simple. It consists of a chunk that contains only the chunk usage identifier and the string for the command, and the newline. One possible valid No-op command is:

```
C NO <nl>
```

Notes to the Implementor

No-ops are included for use in testing and initial synchronization. (The latter use is not mandatory, however. That is, no exchange of No-ops is required at start-up time, but it is conceivable that some implementations might want to do it just for exercise.) They are also traditional.

RFC 929
Proposed Host-Front End Protocol

December 1984

APPENDIX

Per-Protocol Offloading Descriptions

1. Command Level Interface to an Off-loaded TCP

This appendix discusses the use of the commands described in the body of this document to provide an interface between a Host process and an off-loaded interpreter of the DoD's Transmission Control Protocol (TCP). The interface described here is functionally equivalent to the interface found in the MIL-STD 1778 specification of TCP. It is not, however, identical, in that some features of the interface are particularly relevant only in an inboard implementation.

The first section describes the mapping between the interface events of MIL-STD 1778 and the commands and responses of this H-FP, and highlights the unique features of the interface. The next sections discuss the details of each command. These details include the specialized usages of the command and the protocol-idiosyncratic parameters for that command.

1.1. Relation to MIL-STD 1778 Interface

Most of the requests and responses of the TCP interface specified in MIL-STD 1778 are mapped directly to H-FP Commands and responses. The exceptions are noted in the following descriptions.

1.1.1. Requests

Unspecified Passive Open, Fully Specified Passive Open, Active Open, and Active Open with Data requests are all implemented using variations of the Begin command. The distinction between Passive and Active Open is made using the Active/Passive parameter of Begin. The distinction between unspecified and fully specified lies in the presence or absence of the destination address fields. An active open with data is identical to a normal active open, except for the presence of data following the command.

The Send Service Request is implemented using the Transmit command. Special protocol idiosyncratic parameters are provided for Urgent, Push, and changing the ULP timeout action and values. The response to the Transmit command indicates that the appropriate Send call has been made.

RFC 929
Proposed Host-Front End Protocol

December 1984

References

(References [1]-[3] will be available in M. A. Padlipsky's "The Elements of Networking Style", Prentice Hall, 1985.)

[1] Padlipsky, M. A., "The Host-Front End Protocol Approach", MITRE-3996, Vol. III, MITRE Corp., 1980.

[2] Padlipsky, M. A., "The Elements of Networking Style", MB1-41, MITRE Corp., 1981.

[3] Padlipsky, M. A., "A Perspective on the ARPANET Reference Model", MB2-47, MITRE Corp., 1982.

[4] Bailey, G., "Network Access Protocol", S-216,718, National Security Agency Central Security Service, 1982.

[5] Day, J. D., G. R. Grossman, and R. H. Howe, "WWMCCS Host to Front End Protocol", 78012.C-INFE.14, Digital Technology Incorporated, 1979.

There is no corresponding response in the specified TCP interface; its only significance is that the Host can issue another Transmit command.

The Allocate event is a specification feature of MIL-STD 1778 to indicate the willingness of the user to accept incoming data across the interface. However, because this is precisely the type of flow control provided by the Channel level, the Allocate event would be a superfluous mechanism. Thus, there is no direct analogy to that event in the H-FP interface. A Host process indicates its willingness to accept new data by informing the channel via its flow control interface (if it has an explicit one).

Close and Abort are provided by the End command. Close uses the graceful version of the End command, while Abort uses the abrupt version. The response indicates that the End command has been received and the corresponding Close or Abort was issued. There is no corresponding response in the specified TCP interface.

Status is provided by using the query form of the Status command. The response to the Status command contains the information (see below).

1.1.2. Responses

The Open Id response is provided so that the user has a shorthand name by which to refer to the connection. With an outboarded TCP interpreter, there is a one-to-one mapping between TCP connections and H-FP channels. Hence, the Open Id event is not needed, since the channel ID is sufficient to indicate the desired connection.

The Open Failure and Open Success responses are provided using OPE-generated responses to Begin commands (which provide the Active and Passive Service response primitives) issued by the Host. The value of the response code indicates whether the Begin command succeeded or failed, and can be mapped to the appropriate Open Failure or Open Success indication by the Host.

Deliver is provided by having the OPE issue a Transmit command. As mentioned above, the "flow control" between the TCP interpreter and the Host is provided by the Channel layer, so no explicit interface events are needed. The

RFC 929
Proposed Host-Front End Protocol

December 1984

response to the Transmit command indicates the data was received by the Host process. There is no corresponding response in the specified TCP interface.

The Closing and Terminate service responses are provided using the End command. Closing is indicated using the graceful version of the command, while terminate is provided using the abrupt version. The response indicates the End command was received by the Host process. There is no corresponding response in the specified TCP interface.

Status Response is provided by a response to the query version of the Status command. The status information is communicated via protocol-idiosyncratic parameters following the Response code.

Error messages are reported using the spontaneously generated version of the Status command issued by the OPE. The error message is provided in a parameter. The response indicates the error message was received by the Host process. There is no corresponding event in the specified TCP interface.

1.2. The Begin Command

The Begin command is used in TCP in three major ways:

1. To inform the OPE that a process in the Host wishes to open a connection to a particular port on a internet address.
2. To inform the OPE that a process in the Host wishes to be informed when a connection attempt is made to any or to a specific port at this Host's internet address.
3. To inform the Host that a connection attempt to the OPE has arrived, and there was no Begin of the second type (passive open) issued by the Host relevant to that particular port.

1.2.1. Specialized Usage

There are four major aspects to the specialized usage of the Begin command and its parameters. These parameters are:

1. The meaning of the Mediation Level parameter

2. The selection of blocking treatment of Transmit command
3. The meaning of the address components
4. The selection of the TCP Active Open with Data primitive.

The Mediation Level parameter has only two possible values when offloading TCP. These are "9" and "0". The normal usage of an off-loaded TCP uses the value "9", which means the Host is in no way involved in the operation of TCP. The value "0" indicates the Host wishes to negotiate with the TCP options.

The normal TCP Send event is non-blocking. That is, when a user issues the send command, it counts on the reliability services of TCP, and is not explicitly notified when the data has reached the other end of the connection and been properly acknowledged. Hence, the default value for this parameter with TCP is "N". There are some applications where the user may not wish to receive a response to a Transmit command until the data has been acknowledged by the other end of the connection. In these cases, the value "B" should be used for this parameter. If such a feature is not supported by the offloaded TCP interpreter, then it is acceptable to issue a 100 level Conditional acceptance indicating that blocking is not supported, but the Begin command will proceed using non-blocking Transmits.

The primary address components of the local and remote addresses refer to the internet addresses of (or a symbolic Host name for) the respective Hosts. The secondary components refer to the particular sockets at those internet addresses. Normally, the secondary components (ports) are specified numerically. They may, however, be specified by name if the port is a well-known service port. In an Active Begin command, the remote addresses primary and secondary components must be specified. The local address components need not be specified, unless the user wishes to indicate that the connection should be from a particular port or a particular internet address of a multi-homed Host. In a Passive Begin command, the remote addresses are specified only if connection attempts from one particular Host are of interest. The local address secondary component must be used to indicate on which port to perform the Listen.

RFC 929
Proposed Host-Front End Protocol

December 1984

The way the TCP Active Open with data is provided is by including the data with the Begin Command. This data is included in the same Channel level chunk, immediately following the newline. If the data is more than a single chunk can hold, then the multi-chunk command feature of the H-FP must be used.

1.2.2. Protocol-Idiosyncratic Parameters

The protocol-idiosyncratic parameter identified for the TCP interface is the "ULP timeout" information. This information includes whether the offloaded interpreter should abort the connection on a ULP timeout or report it to the inboard user, and also the numerical value of the timeout interval. The format chosen for this parameter is a single letter followed immediately (with no spaces) by an ASCII number. The letter can be either "R" or "A", and indicates that the ULP timeout should cause a report or an abort, respectively. The number is interpreted to be the timeout interval in seconds.

1.2.3. Examples of the Command

An example of an Active Begin command that might be issued by an inboard user Telnet is:

```
C BE TCP A ISIA 9 N 23 ,, 60 R 0 -pi R120 <nl>
```

ISIA is the destination Host, 23 is the well-known port number for Telnet connections, a Begin timeout of 60 seconds was chosen. The desired type of service is to strive for good response time, the transmissions are expected to be in small units, and protocol-idiosyncratic parameter R120 implies that a ULP timeout of 120 seconds should be reported.

An example of a Passive Begin Command that might be issued by an inboard server Telnet is:

```
C BE TCP P ,, 9 N ,, 23 ,, R 0 -pi R120 <nl>
```

The major differences are that no remote address components are specified, and the local secondary address component is identified as the socket on which the Listen is being performed. Also, the default ("infinite") timeout is taken.

1.3. The Transmit Command

The Transmit command is used by the Host process to instruct the off-loaded TCP interpreter to send data to a remote site via the TCP connection associated with the command's channel. It is used by the OPE to deliver incoming data from the connection to the process in the Host.

1.3.1. Specialized Usage

The Transmit command must be capable of providing all the specialized features of the Send and Deliver Event. These special features are Urgent, Push, and modification of the ULP Timeout action and/or interval.

Urgent is a means to communicate that some point upcoming in the data stream has been marked as URGENT by the sender. While the actual Urgent bit travels through the connection out-of-band, it carries a pointer that is related to the sequence numbers of the in-band communication. Hence, the urgency must be indicated in the Transmit command rather than the Signal command.

Push is a feature of the TCP Send Event that is used to indicate that the data in the Transmit command should be sent immediately (within the flow control constraints), rather than waiting for additional send commands or a timeout. Push is indicated in the Transmit Command. The push feature has the same meaning when sent from the OPE to the Host. If the Host implementation does no internal queuing, the flag has no meaning.

The TCP Send event permits the user to modify the "ULP timeout action" and/or the "ULP timeout interval" associated with that connection. When changed, the new values take effect for the remainder of the connection, unless changed later with another Send. This feature is provided in this H-FP using the Transmit Command.

1.3.2. Protocol-Idiosyncratic Parameters

The three features identified above are provided using protocol-idiosyncratic parameters.

The first such parameter is the Urgent parameter. From the point of view of the interface, it is just a flag that indicates the data is urgent (the actual Urgent pointer is a

RFC 929
Proposed Host-Front End Protocol

December 1984

concern of the off-loaded TCP interpreter, which is keeping track of the sequence numbers). When issued by the Host process, the Urgent flag means the stream should be marked. When issued by the OPE, it means the receiver should go to (or remain in) the Urgent receive mode. If the flag is not set in the Transmit issued by the OPE, then the receiver should remain in (or return to) the non-urgent receive mode. The value of this protocol-idiosyncratic parameter is "U" if the Urgent is set, or "N" if it is not set. The default value for this parameter is "N". Since this parameter is the first protocol-idiosyncratic parameter for the Transmit command, it requires no special flag, and can be indicated using the flag -pi.

The second protocol-idiosyncratic parameter is the Push flag. This parameter is only issued by the Host, since there is no Push in the TCP Deliver event. Its value is "P" for push, or "N" for normal. The default value of this parameter is "N". Its control flag is -pu.

The third protocol-idiosyncratic parameter is the ULP timeout action and value parameter. The action part indicates whether the offloaded interpreter should abort the connection on a timeout or report it to the inbound user. The value part is the numerical value of the timeout interval. The format used for this parameter is the same as in the Begin command, which is a single letter followed immediately (with no spaces) by an ASCII number. The letter can be either "R" or "A", and indicates that the ULP timeout should cause a report or an abort, respectively. The number is interpreted to be the timeout interval in seconds. The default interpretation for this parameter is its previous value. The control flag for this parameter is -ul.

1.3.3. Examples of the Command

An example of a Transmit command issued by a Host process is

```
C TR -pi N P R160 <nl> <DATA>
```

where <DATA> is the data contained within the chunk. This command is for a non-urgent but pushed TCP Send event, that also resets the timeout action and interval to Report with a value of 160 seconds. The response mode (i.e., nonblocking) is derived from the Begin command and not effected by transmit

RFC 929
Proposed Host-Front End Protocol

December 1984

An example of a Transmit command issued by the OPE is

```
C TR -pi N <nl> <DATA>
```

where <DATA> is the data contained within the chunk. This command is for a non-urgent delivery (presumably, after a previous Urgent delivery).

1.4. The Condition Command

The Condition command is used to modify the transmission characteristics of the connection. The parameters that make sense to modify with TCP are the Transmit Response discipline, the Type of Service, and the Flow Control Advice.

1.4.1. Specialized Usage

There is no usage of the Condition command with an offloaded TCP interpreter that is particularly specialized.

1.4.2. Protocol-Idiosyncratic Parameters

There are no protocol-idiosyncratic parameters for the condition command for the off-loaded TCP. It would be possible for the ULP timeout action values to be changed with a condition command. However, this is accomplished with the Transmit command, which more closely models the interface specified in MIL-STD 1778. We propose that the condition command not provide this capability.

1.4.3. Examples of the Command

An example of the Condition command to change the flow control advice for a connection is

```
C CO -fc 1 <nl>
```

which indicates that relatively small transmission units are now expected.

RFC 929
Proposed Host-Front End Protocol

December 1984

1.5. The Signal Command

As we currently understand it, TCP's URGENT feature provides an INband signal rather than a true out-of-band signal (and at least one of us deeply regrets this). The actual URGENT bit is sent out-of-band, but it contains an URGENT pointer which relates the URGENT to its position in the data stream. The actual semantics of the URGENT is left to the higher level protocol (e.g., Telnet says to discard all data up to the URGENT pointer). Since the Signal command is allowed to cross a pending Transmit in the H-FP channel, it would be potentially dangerous to implement the interface to TCP URGENT using the Signal command since the wrong sequence number could be used as the urgent pointer. Barring persuasive arguments to the contrary, it is proposed that Signal should not be used with TCP.

1.6. The Status Command

The Status command maps directly into the TCP Status event when issued by a Host process. It is also used for the TCP error event when issued by the OPE. There is currently some question as to how information from lower protocol levels (e.g., ICMP error messages) should be reported to TCP users. When these issues are resolved, there may be other uses for the Status command. We solicit other ideas for the Status command with this report.

1.6.1. Specialized Usage

The major specialized usage of the Status command is to provide the error reporting service. This usage is a form of the Status generated by the OPE.

1.6.2. Protocol-Idiosyncratic Parameters

When used as a TCP Status request (command issued by the Host process), there are no protocol-idiosyncratic parameters associated with the Status command. The OPE response codes the TCP status.

When used as a TCP error report (command issued by the OPE), there is one protocol-idiosyncratic parameter associated with the Status command. It is an error description in the form of a text string. It requires no special control flag since the flag -pi is unambiguous and there are no other protocol-idiosyncratic parameters.

1.6.3. Examples of the Command

An example of the Status command issued by the Host process to request status information is

```
C ST Q <nl>
```

The status information is returned in the response to the status command.

An example of the Status command issued by the OPE to report an error from the TCP interpreter is

```
C ST R -pi "Connection already exists" <nl>
```

which is issued when a TCP open (HEP Begin) is issued to an already opened (foreign) connection.

1.7. The End Command

The End command is used to indicate that TCP services are no longer required. Thus, it can be mapped into either the TCP Graceful Close or the TCP Abort events. It is also used as the TCP Closing response (as contrasted with the response by the OPE to the close command), when issued by the OPE.

1.7.1. Specialized Usage

Because of the nature of the two-way close provided by TCP, there is a possibility that the Host and the OPE wish to gracefully terminate the connection at the same instant. If this happens, then both the Host and the OPE would issue End commands at the same time. To be prepared for this, it is necessary to make this the normal graceful closing sequence. In other words, both the Graceful Close request and the Closing response are mapped to End commands, and the response to one of those commands only indicates that the command has been received and executed, but not that the connection is actually closed. The connection is gracefully closed when both End commands have been issued, and both successful responses have been received.

With an abrupt end, a two-way exchange is not necessary. Only the Host or the OPE need issue it, for the connection to be aborted.

RFC 929
Proposed Host-Front End Protocol

December 1984

1.7.2. Protocol-Idiosyncratic Parameters

There are no protocol-idiosyncratic parameters for the End command used with TCP.

1.7.3. Examples of the Command

An example of the End command used to indicate either a TCP Close request (from the Host process) or TCP Closing response (from the OPE) is

```
C EN G <nl>
```

An example of the End command used as an Abort request (from the Host process) or as a Terminate response is

```
C EN A <nl>
```

2. Command Level Interface to an Off-loaded Telnet

This appendix is provided to discuss the use of the commands described in the body of this document to provide an interface between a Host process and an off-loaded interpreter of the Telnet protocol.

The interface described here is not based on a formal interface. There are several reasons for this, including the lack of a widely accepted standard interface to Telnet, and its headerless nature. Consequently, the interface described here is very similar to the actual Telnet data stream.

2.1. The Begin Command

The Begin command is used with Telnet to initiate Telnet connections.

2.1.1. Specialized Usage

There are three major specialized usages to the Begin command. They are the meaning of the Mediation Level parameter, the way the number of incoming Telnet connections are supported, and the meaning of the secondary address components.

The mediation level is used in Telnet to control which of the various Telnet activities are performed by the OPE, and which are controlled by the Host. It has been determined

that all monitoring of the Telnet Socket should be performed by the OPE. Mediation level 9, which is the default, indicates the Host desires to play no role in Telnet operation. Level 5 means that protocol-idiosyncratic parameters to this Begin command indicate which incoming options the Host wishes to handle; all other options, and all NVT translations, are to be performed by the OPE. Level 0 indicates that the Host will handle all options, while all NVT translations are to be performed in the OPE (see Section B.1.3).

The Host can either accept the connections by fielding OPE generated Begins, or by issuing passive Begins to the OPE. The Host may wish to restrict the number of incoming Telnet connections that it will handle at any particular time. It can do this by rejecting OPE-generated Begins above a certain number, or by limiting the number of Host-issued passive Begins. However, precedence constraints dictate that the Host actually issue additional passive Begins or accept additional Begins from the OPE beyond the maximum number it is normally willing to support, so that high-priority service requests can be accommodated, possibly by preempting lower priority activities.

The secondary address component is used to refer to specific ports. Normally, they are used only when the standard or default ports are not used, such as special purpose applications or testing.

2.1.2. Protocol-Idiosyncratic Parameters

The protocol-idiosyncratic parameters to the Telnet Begin command are the identifiers for the options which the host wishes to negotiate when using mediation level 5. On other mediation levels, these parameters are not used.

2.1.3. Examples of the Command

An example of a passive Begin for an outboard Telnet protocol is:

```
C BE TEL P .. 5 N -fc 0 -pl 9 <nl>
```

Where the parameters are:

```
TEL  Code for the Telnet Protocol
P    Passive Begin
```


RFC 929

December 1984

Proposed Host-Front End Protocol

```
..    Skip the Foreign Address Primary Component
5     Mediation Level is 5
N     Non Blocking Transmits
-fc   Skips over parameters up to Flow Control Advice
S     Small Blocks are appropriate for Telnet
-pi   Skips over parameters to the Protocol Idiosyncratic
      List of Options to be Handled by the Host.
9     Option Code for Line Length Option
```

Here, no remote address component was specified, since the Host will accept connections from any Host. Similarly, no local addresses are specified, since the default well-known socket for this Host is to be used. In this example, the Host specifies it will handle the line length option (number 9). Other options are handled in the OPE.

An example of an active Begin for an outboard Telnet protocol is:

```
C BE TEL A ISIA 5 N -fc 0 -pi 9 <nl>
```

This command is identical to the passive command, except that a remote primary address component is specified to identify the intended Host. No remote secondary component is specified, since the well-known socket at that Host is to be used. No local secondary address components are specified, since the connection can originate from any available socket of the appropriate type selected by the OPE.

2.2. The Transmit Command

The Transmit Command is used to send data across a Telnet connection.

2.2.1. Specialized Usage

The Transmit command is used to transmit data over the Telnet connection. There is one specialized aspect of the Transmit command used with an outboard Telnet interpreter. This is the provision of the Go Ahead feature of Telnet that supports half-duplex devices.

Go Ahead is provided as a protocol idiosyncratic parameter to the Transmit. It is only used if the Host will support it, however. It is our opinion that Go Ahead is probably not a proper thing for the default case.

Go Aheads are a matter between the Host and the terminal. It is difficult to offload the generation of Go Aheads to the OPE, since the OPE is not really cognizant of the semantics of the communication between the Host and the terminal. Hence, the OPE does not know when the Host is done transmitting and willing to pass "the turn" back to the terminal. Similarly when the remote site relinquishes control, the OPE includes Go Ahead in its TR.

We don't believe this Go Ahead problem to be an indictment against outboard processing. It merely illustrates that functionality not found in a Host cannot necessarily be provided by the OPE. Hence, we provide this note to the implementor: if the Host cannot generate the protocol-idiosyncratic Go Ahead parameter, then the DO Suppress Go Ahead must be issued immediately after the connection is established.

2.2.2. Protocol Idiosyncratic Parameters

The protocol idiosyncratic parameter is the Go Ahead indicator. When present, the character "G" is used to mean the Go Ahead can be sent to the other end of the connection, but only after the data associated with that Transmit command is sent. When the character is any other value, or is absent, the Go Ahead should not be sent.

2.2.3. Examples of the Command

An example of the Transmit command is:

```
C TR -pi G <nl> <DATA>
```

With this command, the Go Ahead is passed to the other side after the data is sent.

2.3. The Condition Command

The Condition command is used with Telnet to modify the Transmission characteristics and to enable or disable Telnet options on a Telnet connection.

2.3.1. Specialized Usage

The Condition command takes on specialized usage with Telnet, in addition to its normal usage. It is used to

RFC 929

December 1984

Proposed Host-Front End Protocol

control the option selection and negotiation process, when such selection is performed by the Host (currently, this is done at mediation levels 5 and 1, but not at level 9).

A set of protocol-idiosyncratic parameters has been defined for this purpose. They are based heavily on the Telnet negotiation and subnegotiation mechanisms. For simple negotiations there are two parameters, a negotiation type (from the set {DO, DONT, WILL, WONT}) followed by the code (numeric) or name (symbolic) for the desired option. The codes for the options are identified below. A basic difference between the H-FP interface to Telnet and the internal Telnet protocol is that additional parameters are included with the request (DO or WILL). The Telnet protocol subnegotiation is used internally to communicate that information in the Telnet data stream. Option-specific, protocol-idiosyncratic parameters are used for these additional parameters.

Both the Host and the OPE can issue these Condition commands. When issued by the Host, it means the user wishes to enable or disable a particular option. The OPE proceeds to issue the appropriate negotiation commands (i.e., IAC <DO> <code>) in the Telnet data stream. When the results of the option negotiation are available, a response is generated by the OPE. For the types DO and WILL, a 000 Response indicates the appropriate acceptance (WILL or DO, respectively). A nonzero Response code may indicate negotiation failure or negotiation rejection (among other things). For the types DONT and WONT, a 000 Response indicates the option will be disabled. A negotiation rejection should not be expected in those cases.

When the Condition command is issued by the OPE, it means the other end of the connection is negotiating a change. Here the response from the Host indicates the Host's desired action for the option negotiation. Again, valid requests to disable options (DONT and WONT requests) should always get a 000 Response.

2.3.2. Protocol-Idiosyncratic Parameters

There are two protocol-idiosyncratic parameters for primary negotiation using the Condition command. These are the negotiation type and the option code. The negotiation type is one of the set of {DO, DONT, WILL, WONT}. The option code is a numeric value used to identify the particular

option being negotiated. The values for these codes are indicated here, but are identical to the codes used in the actual Telnet negotiation. The codes are:

Option Name	Option Code	Short Name
Transmit Binary	0	Binary
Echo	1	Echo
Suppress Go-Ahead	3	SuppressGA
Approximate Message Size	4	NAMS
Status	5	Status
Timing Mark	6	TimingMark
RCTE	7	RCTE
Line Length	8	LineLength
Page Size	9	PageSize
Carriage Return Disp	10	CRDisp
Horizontal Tabstops	11	HTabStops
Horizontal Tab Disp	12	HTabDisp
Formfeed Disposition	13	FFDisp
Vertical Tabstops	14	VTabStops
Vertical Tab Disposition	15	VTabDisp
Linefeed Disposition	16	LFDisp
Extended ASCII	17	ExASCII
Logout	18	Logout
Data Entry Terminal	20	DET
Terminal Type	24	TermType
Extended options list	255	ExOptions

Options not listed here may of course be used. The code number should be the same as the option code used in Telnet negotiation.

2.3.2.1. Simple Options

Options that do not require additional parameters use the simple negotiation mechanisms described briefly above and in greater detail in the Telnet documentation. No additional parameters are required. These options include the Transmit Binary, Echo, Suppress Go Ahead, Status, Timing Mark, and Logout options.

2.3.2.2. Approximate Message Size Option

The Approximate Message Size option requires two parameters. The first indicates whether the approximate message size being negotiated applies to the local or the remote end of the connection. DS means the size applies

to the sender of the command (i.e., if the Host issues the command, DS means the local end of the connection; if issued by the OPE, DS means the remote end of the connection). DR means the size applies to the receiver of the command (i.e., if the Host issues the command, DR means the remote end; if issued by the OPE, DR means the local end of the connection). This convention is consistent with the Telnet subnegotiation mechanisms. The second character is an ASCII encoded numeric value, which is a character count of the message size.

2.3.3. Line Width and Page Size Options

The Line Width and Page Size Options require two additional parameters. The first indicates whether the line width or page size being negotiated applies to the local or the remote end of the connection, and uses the DS and DR convention described above. The second parameter is an ASCII encoded numeric value, which is interpreted as follows (assuming the Condition command was issued by the Host):

- | | |
|----------|---|
| 0 | The Host requests that it handle length or size considerations for the direction indicated by the first parameter. |
| 1 to 253 | The Host requests that the remote end handle the size or length considerations for the direction indicated by the first parameter, but suggests that the value indicated be used as the size or length. |
| 254 | The Host requests that the remote end handle the size or length considerations for the direction indicated by the first parameter, but suggests that the size or length be considered to be infinity. |
| 255 | The Host requests that the remote end handle the tabstop considerations, and suggests nothing about what the value should be. |

If the Condition command is issued by the OPE, then the roles of the Host and the remote end are reversed.

2.3.4. Tabstop Options

The Horizontal and Vertical Tabstops options require two option specific parameters. The first is either DR or DS, as was described previously. The second is a list of one or more ASCII encoded numeric values separated by spaces which, assuming the Condition command is issued by the Host, are individually interpreted as:

- 0 The Host requests that it handle tabstops for the direction indicated by the first parameter.
- 1 to 250 The Host requests that the remote end handle the tabstop considerations for the direction indicated by the first parameter, but suggests that the value(s) indicated should be used as the tabstops.
- 255 The Host requests that the remote end handle the tabstop considerations for the direction indicated by the first parameter, and suggests nothing about what the value should be.

If the Condition command is issued by the OPE, then the roles of the Host and the remote end are reversed.

2.3.5. Character Disposition Options

The Carriage Return Disposition option, the Horizontal Tab Disposition option, the Formfeed Disposition option, the Vertical Tab Disposition option, and the Linefeed Disposition option are all considered character disposition options from the perspective of H-FP. Two option-specific parameters are required for the character disposition options. The first is the DR or DS code, which was described previously. The second is a single ASCII encoded numeric value, which is interpreted as (assuming that the Host issued the Condition command):

- 0 The Host requests that it handle the character disposition for this connection.
- 1 to 250 The Host suggests that the remote end handle the character disposition considerations, but suggests that the value indicated should be taken as the number of nulls which should be

RFC 929
Proposed Host-Front End Protocol

December 1984

inserted in the data stream following the particular format character being subnegotiated.

- 251 The Host suggests that the remote end handle the character disposition considerations, but recommends that it replace the character with some simplified character similar to but not identical with it (e.g., replace a tab with a space, or a formfeed with a newline).
- 252 The Host suggests that the remote end handle the character disposition considerations, but recommends that it discard the character.
- 253 The Host suggests that the remote end handle the character disposition, but recommends that the effect of the character be simulated using other characters such as spaces or linefeeds.
- 254 The Host suggests that the remote end handle the character disposition considerations, but recommends that it wait for additional data before sending more data.
- 255 The Host suggests that the remote end handle the tabstop considerations, and suggests nothing about what the value should be.

Some of the codes between 251 and 254 are not used with some character disposition options. Refer to the ARPANET documentation for additional details.

If the Condition command is issued by the OPE, then the roles of the Host and the remote end are reversed.

2.3.5.1. RCTE Option

The Remote Controlled Transmission and Echoing option requires parameters to indicate the sets of break characters and transmit characters. There are two option-idiosyncratic parameters for RCTE. The first is a list of the character classes that make up the set of break characters, as defined in the RCTE documentation. The second is a list of character classes that make up the set of transmit characters, as defined in the RCTE documentation. Since the two classes are optional and

can be of arbitrary length, it is necessary to precede each list with a -bc (break characters) or -tc (transmit characters). The character classes are defined as

- 1 Upper Case Letters A through Z
- 2 Lower Case Letters a through z
- 3 Digits 0 through 9
- 4 Format effectors <BS> <CR> <LF> <FF> <HT> <VT>
- 5 Non-format control codes, plus <ESC> and
- 6 Punctuation . , ; : ? !
- 7 Grouping { [(< >)] }
- 8 Misc ' ` " / \ % @ \$ & + - * = ^ _ | ~
- 9 <space>

2.3.5.2. Extended Option List

The Extended Option List option requires a parameter to carry the number of the option on the extended list. There is thus one option specific parameter to the Condition command when used for this purpose, which is the number of the option on the extended option list. It can be expressed in ASCII using an octal, decimal, or hexadecimal format.

2.3.5.3. Terminal Extension Options

The Extended ASCII, SUPDUP, and Data Entry Terminal options of Telnet were all attempts to extend the basic capabilities of the Telnet data stream beyond the simple, scroll mode terminal model that was the basis of the original Telnet design.

All of these options have limitations to their effectiveness. The Extended ASCII option lacks a standardized interpretation of the bit patterns into extended ASCII characters. The SUPDUP effort was actually an independent mode where a different virtual terminal protocol was used, and the option was there merely to switch to and from this protocol. The Data Entry Terminal option requires the excessive overhead of subnegotiation for each use of extended features. All of these options lack the more valuable asset of widespread implementation and use.

The way these options should be handled is not detailed in this appendix. It is clear that the Condition command could be used for initiating and terminating the use of

these options. The actual transmission of characters related to the extended terminal features should be provided by the Transmit command, either as part of the normal Host-to-OPE data stream or by using protocol-idiosyncratic parameters.

A more recent option, the Terminal Type option, should be mentioned here. It permits one end of a connection to request information about the terminal at the other end or send information about the terminal at the local end. This is convenient for systems that provide a wide variety of terminal support, but it clearly does not follow the model of reducing the M&N problem by use of a virtual terminal. Its use is very straightforward in the H-FP context. It only requires sending the terminal type to the other end, and activating the Binary Transmission Option.

2.3.5.4. Status Option

The Status option is enabled using the negotiation mechanism of Telnet. However, the means to transfer status information between OPE and the Host is provided via the Status command. Therefore, details of status negotiation are irrelevant to the interface to the outboard Telnet.

2.3.6. Examples of the Command

The following example shows the command issued by a Host to the OPE, requesting that the OPE negotiate with the other side so that remote echo is performed.

```
C CO -pi DO 1 <nl>
```

The numeral 1 is the option code for ECHO from the table above. All of the simple options listed above use this same basic format.

The options with additional parameters use straightforward extensions of this syntax. For example, a possible usage of Condition by the Host to set the approximate message size is:

```
C CO -pi DO 4 DS 1024
```

RFC 929
Proposed Host-Front End Protocol

December 1984

2.5. The Status Command

The Status command is used with Telnet to obtain information about the Telnet connection and the options in effect.

2.5.1. Specialized Usage

The Status command has one specialized aspect when used to interface to an outboard Telnet interpreter. That is to send and receive the Telnet Protocol status request subnegotiation message to and from the data stream. In order to invoke the status command for this purpose, however, the user must have previously issued the Condition Status command, which causes the ability to request status to be negotiated. The OPE, when it receives a valid Status request command, immediately responds to the user indicating the status. The OPE can issue a status to request the Host's negotiated positions.

2.5.2. Protocol-Idiosyncratic Parameters

There are no protocol-idiosyncratic parameters to the Status query command. The Status Response command has a single protocol-idiosyncratic parameter. It is an ASCII string containing the status of the various options (not at their default values).

2.5.3. Examples of the Command

An example of a Status Query command is:

```
C ST Q
```

An example of a Status Response command is:

```
F ST R "WILL ECHO DO SUPPRESS-GO-AHEAD  
L WILL STATUS DO STATUS" <nl>
```

In the previous example, note the opening quote is in the first chunk, and the closing quote is in the last chunk. This technique permits parameters to span chunk boundaries.

The 4 is the Option Code for the Approximate Message Size option, the DS indicates that Host's message size should be set, and 1024 is the desired size.

2.4. The Signal Command

The Signal command is used with Telnet to provide the Telnet Interrupt Process and Abort Output services.

2.4.1. Specialized Usage

The Signal command is used with an outboard Telnet interpreter to interface to the Telnet synch mechanism. This mechanism is used with a protocol-idiosyncratic parameter, which indicates what particular command is being "synched." It is expected that normally, this Signal mechanism will only be used with the Interrupt Process and Abort Output Telnet signals. When the Signal command is issued by the Host, it goes through the Channel (out-of-band) to the OPE, where the Telnet interpreter issues the corresponding Telnet signal and synch sequence. When such a sequence is received by the OPE, it immediately issues a Signal to the Host. It is expected that a Host or OPE would not, in general, reject the Signal command unless it is badly formed.

2.4.2. Protocol-Idiosyncratic Parameters

The Telnet protocol-idiosyncratic parameter used with the Signal command identifies which Telnet signal is begin issued. Normally, it would have the value of either "IP" or "AO", for Interrupt Process or Abort Output. If absent, the default value is "IP".

2.4.3. Examples of the Command

An example of a Telnet Signal Command (in this case, to send an Interrupt Process signal) is:

```
C SI IP <nl>
```

RFC 929
Proposed Host-Front End Protocol

December 1984

2.6. The End Command

The End command is used to terminate the Telnet connection, either gracefully or abruptly.

2.6.1. Specialized Usage

The graceful termination of a Telnet requires End commands to be issued by both the Host and the OPE. This specialized usage is identical to that of the outboard TCP interface, however.

2.6.2. Examples of the Command

An example of the graceful End command is:

C EN G <nl>

The abrupt End command is similar.

2.7. The No-op Command

The No-op command is used with Telnet so the Host can determine if the OPE is active, and vice versa.

2.7.1. Specialized Usage

The No-op command has one specialized usage when offloading Telnet. This is to provide the Telnet Are You There (AYT) feature. When an (AYT) message is received by the OPE, it issues a No-op command to the Host. Upon receiving the response from the Host, the appropriate response is sent back in the data stream.

2.7.2. Protocol Idiosyncratic Parameters

There are no protocol-idiosyncratic parameters to the No-op command.

2.7.3. Examples of the Command

An example of the No-op command is:

C NO <nl>

RFC 929
Proposed Host-Front End Protocol

December 1984

3. FTP Offloading

TBS

4. Mail Offloading

TBS

5. Whatever Offloading

TBS

Where TBS nominally = To Be Supplied, but really means: We'll argue through these once we get sufficiently positive feedback on the others (and on the H-FP as a whole).

Network Working Group
Request for Comments: 792

J. Postel
ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION

Introduction

The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling fragment zero of fragmented datagrams. (Fragment zero has the fragment offset equal zero).

[Page 1]

RFC 792

September 1981

Message Formats

ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is a ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and must be zero when sent, but receivers should not use these fields (except to include them in the checksum). Unless otherwise noted under the individual format descriptions, the values of the internet header fields are as follows:

Version

4

IHL

Internet header length in 32-bit words.

Type of Service

0

Total Length

Length of internet header and data in octets.

Identification, Flags, Fragment Offset

Used in fragmentation, see [1].

Time to Live

Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse.

Protocol

ICMP = 1

Header Checksum

The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

[Page 2]

September 1981
RFC 792

Source Address

The address of the gateway or host that composes the ICMP message. Unless otherwise noted, this can be any of a gateway's addresses.

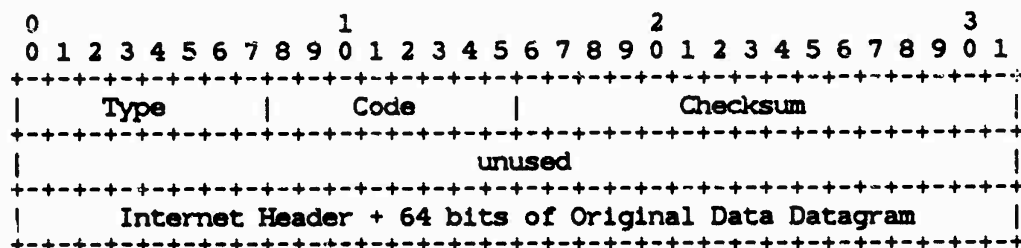
Destination Address

The address of the gateway or host to which the message should be sent.

September 1981

RFC 792

Destination Unreachable Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

3

Code

- 0 = net unreachable;
- 1 = host unreachable;
- 2 = protocol unreachable;
- 3 = port unreachable;
- 4 = fragmentation needed and DF set;
- 5 = source route failed.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original

[Page 4]

September 1981
RFC 792

datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If, according to the information in the gateway's routing tables, the network specified in the internet destination field of a datagram is unreachable, e.g., the distance to the network is infinity, the gateway may send a destination unreachable message to the internet source host of the datagram. In addition, in some networks, the gateway may be able to determine if the internet destination host is unreachable. Gateways in these networks may send destination unreachable messages to the source host when the destination host is unreachable.

If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.

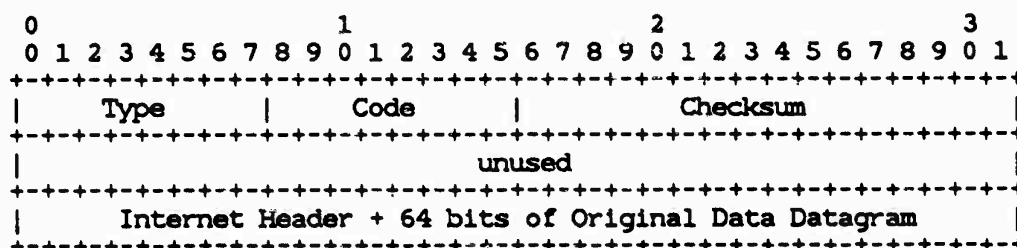
Another case is when a datagram must be fragmented to be forwarded by a gateway yet the Don't Fragment flag is on. In this case the gateway must discard the datagram and may return a destination unreachable message.

Codes 0, 1, 4, and 5 may be received from a gateway. Codes 2 and 3 may be received from a host.

September 1981

RFC 792

Time Exceeded Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

11

Code

0 = time to live exceeded in transit;

1 = fragment reassembly time exceeded.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If the gateway processing a datagram finds the time to live field

[Page 6]

September 1981
RFC 792

is zero it must discard the datagram. The gateway may also notify the source host via the time exceeded message.

If a host reassembling a fragmented datagram cannot complete the reassembly due to missing fragments within its time limit it discards the datagram, and it may send a time exceeded message.

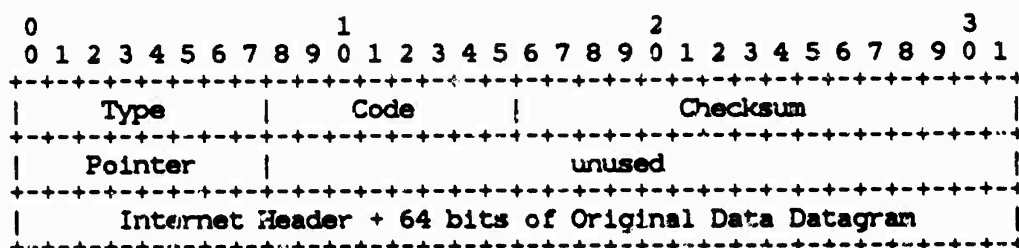
If fragment zero is not available then no time exceeded need be sent at all.

Code 0 may be received from a gateway. Code 1 may be received from a host.

September 1981

RFC 792

Parameter Problem Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

12

Code

0 = pointer indicates the error.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Pointer

If code = 0, identifies the octet where an error was detected.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

[Page 8]

September 1981
RFC 792

Description

If the gateway or host processing a datagram finds a problem with the header parameters such that it cannot complete processing the datagram it must discard the datagram. One potential source of such a problem is with incorrect arguments in an option. The gateway or host may also notify the source host via the parameter problem message. This message is only sent if the error caused the datagram to be discarded.

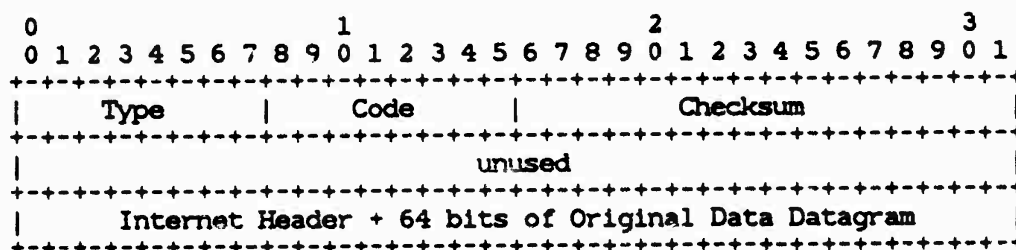
The pointer identifies the octet of the original datagram's header where the error was detected (it may be in the middle of an option). For example, 1 indicates something is wrong with the Type of Service, and (if there are options present) 20 indicates something is wrong with the type code of the first option.

Code 0 may be received from a gateway or a host.

September 1981

RFC 792

Source Quench Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

4

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

A gateway may discard internet datagrams if it does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network. If a gateway

[Page 10]

September 1981
RFC 792

discards a datagram, it may send a source quench message to the internet source host of the datagram. A destination host may also send a source quench message if datagrams arrive too fast to be processed. The source quench message is a request to the host to cut back the rate at which it is sending traffic to the internet destination. The gateway may send a source quench message for every message that it discards. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the gateway. The source host can then gradually increase the rate at which it sends traffic to the destination until it again receives source quench messages.

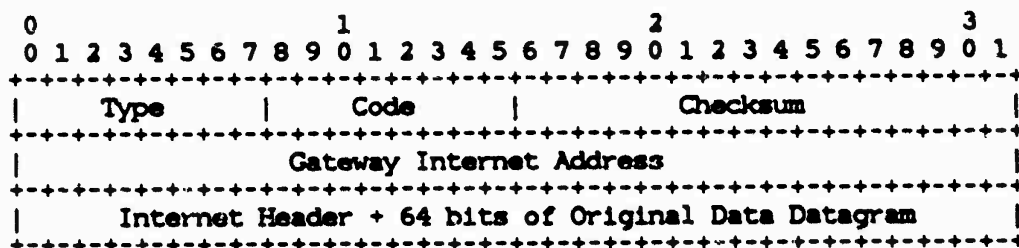
The gateway or host may send the source quench message when it approaches its capacity limit rather than waiting until the capacity is exceeded. This means that the data datagram which triggered the source quench message may be delivered.

Code 0 may be received from a gateway or a host.

September 1981

RFC 792

Redirect Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

5

Code

0 = Redirect datagrams for the Network.

1 = Redirect datagrams for the Host.

2 = Redirect datagrams for the Type of Service and Network.

3 = Redirect datagrams for the Type of Service and Host.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Gateway Internet Address

Address of the gateway to which traffic for the network specified in the internet destination network field of the original datagram's data should be sent.

[Page 12]

September 1981
RFC 792

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

The gateway sends a redirect message to a host in the following situation. A gateway, G1, receives an internet datagram from a host on a network to which the gateway is attached. The gateway, G1, checks its routing table and obtains the address of the next gateway, G2, on the route to the datagram's internet destination network, X. If G2 and the host identified by the internet source address of the datagram are on the same network, a redirect message is sent to the host. The redirect message advises the host to send its traffic for network X directly to gateway G2 as this is a shorter path to the destination. The gateway forwards the original datagram's data to its internet destination.

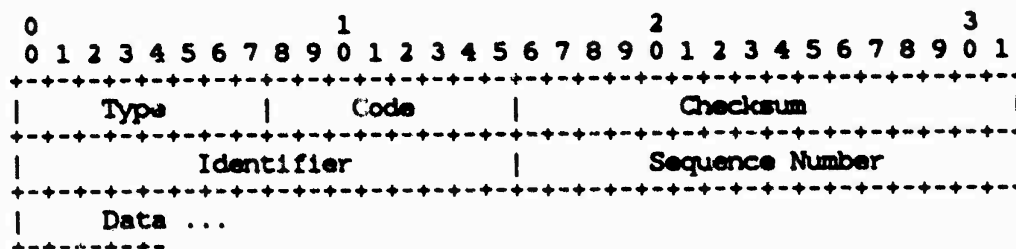
For datagrams with the IP source route options and the gateway address in the destination address field, a redirect message is not sent even if there is a better route to the ultimate destination than the next address in the source route.

Codes 0, 1, 2, and 3 may be received from a gateway.

September 1981

RFC 792

Echo or Echo Reply Message



IP Fields:

Addresses

*The address of the source in an echo message will be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

IP Fields:

Type

8 for echo message;

0 for echo reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching echos and replies, may be zero.

Sequence Number

[Page 14]

September 1981
RFC 792

If code = 0, a sequence number to aid in matching echos and replies, may be zero.

Description

The data received in the echo message must be returned in the echo reply message.

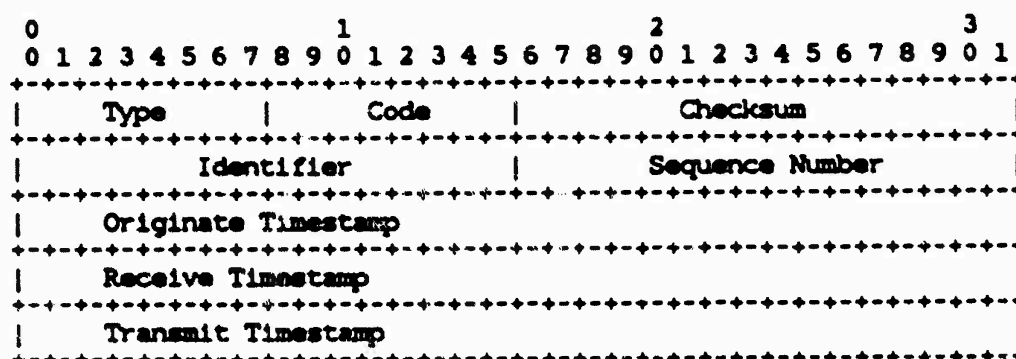
The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.

Code 0 may be received from a gateway or a host.

September 1981

RFC 792

Timestamp or Timestamp Reply Message



IP Fields:

Addresses

The address of the source in a timestamp message will be the destination of the timestamp reply message. To form a timestamp reply message, the source and destination addresses are simply reversed, the type code changed to 14, and the checksum recomputed.

IP Fields:

Type

13 for timestamp message;

14 for timestamp reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Identifier

[Page 16]

September 1981
RFC 792

If code = 0, an identifier to aid in matching timestamp and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching timestamp and replies, may be zero.

Description

The data received (a timestamp) in the message is returned in the reply together with an additional timestamp. The timestamp is 32 bits of milliseconds since midnight UT. One use of these timestamps is described by Mills [5].

The Originate Timestamp is the time the sender last touched the message before sending it, the Receive Timestamp is the time the echoer first touched it on receipt, and the Transmit Timestamp is the time the echoer last touched the message on sending it.

If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value.

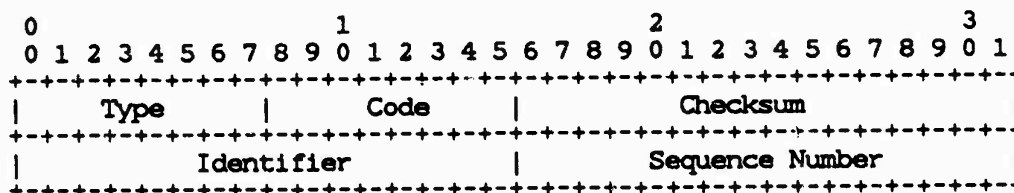
The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

September 1981

RFC 792

Information Request or Information Reply Message



IP Fields:

Addresses

The address of the source in a information request message will be the destination of the information reply message. To form a information reply message, the source and destination addresses are simply reversed, the type code changed to 16, and the checksum recomputed.

IP Fields:

Type

15 for information request message;

16 for information reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching request and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching request and replies, may be zero.

[Page 18]

September 1981
RFC 792

References

- [1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.
- [2] Cerf, V., "The Catenet Model for Internetworking," IEN 48, Information Processing Techniques Office, Defense Advanced Research Projects Agency, July 1978.
- [3] Strazisar, V., "Gateway Routing: An Implementation Specification", IEN 30, Bolt Beranek and Newman, April 1979.
- [4] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.
- [5] Mills, D., "DCNET Internet Clock Service," RFC 778, COMSAT Laboratories, April 1981.

REFERENCES

SECTION 7. REFERENCES

1. *RFC 945, A DoD Statement on the NRC Report*, University of Southern California, Information Sciences Institute, Marina del Rey, CA, May 1985.
2. *RFC 942, Transport Protocols for Department of Defense Data Networks*, Report to the Department of Defense and the National Bureau of Standards, National Research Council, Committee on Computer-Computer Communication Protocols, Washington, DC, February 1985.

RFC 792

September 1981

Summary of Message Types

- 0 Echo Reply
- 3 Destination Unreachable
- 4 Source Quench
- 5 Redirect
- 8 Echo
- 11 Time Exceeded
- 12 Parameter Problem
- 13 Timestamp
- 14 Timestamp Reply
- 15 Information Request
- 16 Information Reply

[Page 20]

September 1981
RFC 792

Description

This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.